```
% lexsemsub.pl
% lexsemsub.pat
% revised  March 17, 2000
%           LEXICON OF SUBSTANCES AND STRUCTURES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:-multifile(phrase/5).
:-multifile(wdef/3).
:-unknown(_,fail).
phrase('[',protein, ['[',gamma,']','-',aminobutyric, acid, a], 'GA
BAA', r).  % ?
phrase('[',smallmolecule, ['[',zeta,']',1, subunit], '[zeta]1 subu
nit', r).  % ?
phrase(116, protein,[116,'-',kd,fyn,'-',associated,protein],'116-k
D Fyn-associated protein',r).
phrase(116, protein,[116,'-',kd,protein], '116-kd protein',r).
phrase(3,protein, [3,'-',kinase,'-',akt],  '3-kinase-Akt',r).
phrase(ability, affirmation, [ability, to], [], r).
phrase(agc,protein, [agc, protein, kinases], 'AGC', r).
phrase(akt,protein, [akt, mutant], 'Akt mutant', r).
phrase(alternative,substance,[alternative,ntf],'alternative NTF',r
).
phrase(antibody, protein,[antibody,to,phosphotyrosine],'anti-phosp
hotyrosine',r).
phrase(antigen, complex,[antigen,receptor], 'antigen receptor',r).
phrase(ap, protein, [ap,'-',1],'AP-1',r).
phrase(aspargine,site,[aspargine,'-',141],'aspargine-141',r).
phrase(b, cell, [b,cell], 'B cell', r).
phrase(b, cell, [b,cells], 'B cell', r).
phrase(b, species,[b,lymphoblastoid,cells], 'B lymphoblastoid cell
s',r).
phrase(b,cell,[b,lymphoblastoid,cells], 'B lymphoblastoid cells',r
).
phrase(b7, protein, [b7,'-','1'], 'B7-1',r).
phrase(bcl,protein,[bcl,'-',2],'Bcl-2',r).
phrase(c, protein, [c,'-',jun] , 'c-Jun',r).
phrase(camk, protein, [camk, iv], 'CaMK IV',r).
phrase(casp, protein,[casp,'-',3], 'caspase-3',r).
phrase(caspase,protein,[caspase,'-',3,family,protease], 'caspase-3
 family protease',r).
phrase(caspase,protein,[caspase,'-',3,precursor],'caspase-3 precur
sor',r).
phrase(caspase,protein,[caspase,'-',3],'caspase-3',r).
phrase(caspase,protein,[caspase,-,3],'caspase-3',r).
```

Appendix A

```
phrase(caspase,protein,[caspase,'-',6],'caspase-6',r).
phrase(caspase,protein,[caspase,'-',7],'caspase-7',r).
phrase(catalytic, domain, [catalytic, domain], 'catalytic domain',
r).
phrase(cleavage,site,[cleavage,site],'cleavage site',r).
phrase(cleavage,substance,[cleavage,products],'cleavage products',
r).
phrase(cooh,substance, [cooh,'-',terminal,fragment], 'COOH-termina
l fragment',r).
phrase(crk,protein,[crk,proteins], 'crk proteins',r0.
phrase(crkl, complex,[crkl,'-',c3g,complex],'crkl-c3g complex',r).
phrase(dcp,protein,[dcp,-,1],'DCP-1',r).
phrase(did, negation, [did, not], not, r).
phrase(ebv,species, 'Epstein-Barr virus',r).
phrase(epstein, species, [epstein,'-',barr,virus],'Epstein-Barr vi
rus',r).
phrase(familial,disease,[familial,alzheimer,'''',s,disease],'famil
ial Alzheimer''''s disease',r).
phrase(gene, gene, [gene,encoding,interleukin,'-',2],'gene encodin
g interleukin-2', r).
phrase(gst, protein, [gst,'-','fyn','-',sh2], 'GST-Fyn-SH2',r).
phrase(gst, protein, [gst,'-','fyn','-',sh3], 'GST-Fyn-SH3',r).
phrase(gtp, complex,[gtp,exchange,of,rap1],'GTP exchange of Rap1',
r).
phrase(guanidine, protein, [guanidine,nucleotide,'-',releasing,fac
tor,c3g],'guanidine nucleotide-releasing factor C3G',r).
phrase(guanidine, smallmolecule, [guanidine,nucleotide],'guanidine
 nucleotide',r).
phrase(guanosine, smallmolecule,[guanosine,triphosphate],'guanosin
e triphosphate',r).
phrase(guanosine,smallmolecule,[guanosine,diphosphate],'guanosine
diphosphate',r).
phrase(h4,cell,[h4,cell,line], 'H4 cell line',r).
phrase(h4,cell,[h4,human,neuroglioma,cells], 'H4,human,neuroglioma
,cells',r).
phrase(ha, protein, [ha, '-', '[',delta,']',phpkb],'HA-[Delta]PHPK
B',r).
phrase(hla, protein, [hla,'-',dr7], 'HLA-DR7',r).
phrase(i, protein, [i, '[',kappa, ']',b,'-','[',beta,']'],   'I[ka
ppa]B-[beta]',r).
phrase(i,protein, [i, '[',kappa, ']',b,'-','[',alpha,']'], 'I[kap
pa]B-[alpha]',r).
phrase(i,protein, [i, '[',kappa, ']',b], 'I[kappa]B',r).
```

```
phrase(ice,protein,[ice,'/',ced,'-',3],'ICE/Ced-3',r).
phrase(il, gene, [il,'-',2,gene], 'gene encoding interleukin-2', r
).
phrase(il, protein, [il,'-',2], 'interleukin-2',r).
phrase(in, interm, [in, the, case, of],[], r).
phrase(in,state,[in,the,anergic,state], inactive,r).
phrase(inducible, cell, [inducible,h4,cell], 'inducible H4 cell',r
).
phrase(interleukin, protein, [interleukin,'-',2],r).
phrase(interleukin, protein,[interleukin, '-', 3], 'interleukin-3
',r).
phrase(interleukin,protein,[interleukin,'-',1,beta,converting,enzy
me], 'interleukin-1 beta converting enzyme',r).
phrase(jurkat, cell, [jurkat, cell], 'Jurkat cell', r).
phrase(jurkat, cell, [jurkat, cells], 'Jurkat cell', r).
phrase(kif3a,protein,[kif3a,'/',3,b],'KIF3A/3B',r).
phrase(lbl, cell, [lbl,'-',drf, cells], 'LBL-DR7 cells',r).
phrase(lbl,cell,[lbl,'-',dr7,cells],'LBL-DR7 cells',r).
phrase(let, protein, [let,'-',23], 'Let-23', r).
phrase(may, probability,[may, be], possible, r).
phrase(myc, protein, [myc, '-', p70s6kd3e], 'Myc-p70s6kD3E',r).
phrase(myc, protein, [myc, '-', pdk1], 'Myc-PDK1',r).
phrase(myc,protein,[myc,'-',p70s6k],'Myc-p70s6k',r).
phrase(myc,protein,[myc,'-',p70s6ke389d3e], 'Myc-p70s6kE389D3E',r)
.
phrase(myr, protein,[myr,'-',akt], 'Myr-Akt',r).
phrase(n,protein, [n,'-',methyl,'-',d,'-',aspartate, receptor], 'N
MDAR', r).
phrase(n,protein, [n,'-',methyl,'-',d,'-',aspartate], 'NMDA').
phrase(native, cell, [native,h4,cell],'native H4 cell',r).
phrase(nf, protein, [nf,'-','[',kappa,']',b], 'NF-[kappa]B',r).
phrase(nh2, site, [nh2,'-',terminal], 'NH2-terminal',r).
phrase(nh2,substance,[nh2,'-',terminal,fragment], 'NH2-terminal fr
agment',r).
phrase(nih, cell,[nih,'-',3,t3,fibroblasts], 'NIH-3T3 fibroblasts'
, r).
phrase(nih,cell,[nih,'-','3t3', fibroblasts],'NIH-3T3 fibroblasts'
,r).
phrase(normal,substance,[normal,ntf],'normal NTF',r).
phrase(nuclear, protein, [nuclear, factor, kappa, b],'NF-[kappa]B'
, r).
phrase(p150Glued,protein,[p150Glued,-,arp1],'p150Glued-Arp1',r).
phrase(phosphate,phosphorylate2, [phosphate, incorporated, into],
```

```
phosphorylate,r).

phrase(phosphatidylinositol, smallmolecule,[phosphatidylinositol,1
,',',4,',',5,'-',triphosphate], 'phosphatidylinositol 1,4,5-tripho
sphate',r).
phrase(phosphoinositide, protein, [phosphoinositide,'-',dependent,
 protein, kinase], 'PDK1',r).
phrase(phospholipase, protein, [phospholipase,c,'-',1],'phospholip
ase C-1', r).
phrase(poly,protein,[poly,'(',adp,'-',ribose,')',polymerase],'poly
(ADP-ribose) polymerase',r).
phrase(polyvinylidene, structure,[polyvinylidene, difluoride, memb
ranes],'polyvinylidene difluoride membranes',r).
phrase(presenilin, protein,[presenilin,1],'presenilin 1',r).
phrase(presenilin,protein,[presenilin,2],'presenilin,2',r).
phrase(productively, state,[productively,stimulated],active,r).
phrase(protein, protein, [protein,tyrosine,kinase],'protein tyrosi
ne kinase', r).
phrase(protein,protein,[protein,kinase,c],'protein kinase C',r).
phrase(ps2,substance,[ps2,'-',ctf],'presenilin 2 COOH-terminal fra
gment',r).
phrase(ps2,substance,[ps2,cleavage,fragment], 'presenilin 2 cleava
ge fragment', r).
phrase(pvdf, structure, [pvdf, membranes],'polyvinylidene difluori
de membranes',r).
phrase(raf, protein, [raf,'-',1], 'Raf-1', r).
phrase(raf,protein,[raf,'-',1], 'Raf-1',r).
phrase(rap1,complex,[rap1,'-',gtp], 'Rap1-GTP',r).
phrase(requirement, need2, [requirement, for], need,r).
phrase(ser, smallmolecule, [ser, 19], 'Ser 19',r).
phrase(ser, smallmolecule, [ser, 23], 'Ser 23',r).
phrase(serine, substance, [serine, residues], 'serine residues', r
).
phrase(src, domain, [src, homology, 2], 'Src homology 2',r).
phrase(src, domain, [src, homology, 3], 'Src homology 3',r).
phrase(srebp,protein,[srebp,'-',1], 'sterol-regulatory element bin
ding protein 1',r).
phrase(srebp,protein,[srebp,'-',2], 'sterol-regulatory element bin
ding protein 2',r).
phrase(sterol,protein,[sterol,'-',regulatory,element,binding,prote
in,1],'sterol-regulatory element binding protein 1',r).
phrase(sterol,protein,[sterol,'-',regulatory,element,binding,prote
in,2],'sterol-regulatory element binding protein 2',r).
```

```
phrase(t, cell, [t,'-',dr7], 't-DR7',r).
phrase(t, cell, [t,'-',drt,'/',b7,'-',1],'t-DR7/B7-1',r).
phrase(t, cell, [t,cell], 'T cell',r).
phrase(t, cell, [t,cells], 'T cell',r).
phrase(t, complex,[t,'-',cell,receptor],'T-cell receptor',r).
phrase(t,cell,[t,'-',dr7, cells],'t-DR7 cells',r).
phrase(t,cell,[t,'-',dr7,'/',b7,'-',1], 't-DR7/B7-1',r).
phrase(t,complex,[t,'-',cell,antigen,receptor],'T-cell antigen rec
eptor',r).
phrase(threonine, aminoacid, [threonine, 229], 'threonine 229', r)
.
phrase(transcription, protein,[transcription, factor], 'transcript
ion factor',r).
phrase(trypan,smallmolecule,'trypan blue',r).
phrase(wt,protein, [wt, akt], 'WT Akt',r).
phrase(zap, protein, [zap,'-',70],'ZAP-70',r).
phrase(zdevd,smallmolecule,[zdevd,'-',fmk],'zDEVD-fmk',r).
phrase(il, protein,[il,'-',3],' interleukin-3',r).
wdef(ab, complex, antibody).
wdef(actin,protein,actin).
wdef(activated, state, active).
wdef(active, state, active).
wdef(ad,disease,'Alzheimer''''s disease').
wdef(agc,protein, 'AGC').
wdef(akt, protein, 'AKT').
wdef(anergic,state,inactive).
wdef(anergic,state,inactive).
wdef(anergy,state,inactive).
wdef(antibody,complex,antibody).
wdef(antigen, substance, antigen).
wdef(aop, protein, 'Aop').
wdef(apoptosis,process,apoptosis).
wdef(bad, protein, 'BAD').
wdef(c3g, protein, 'C3G').
wdef('ca2+', smallmolecule,'Ca2+').
wdef(cas, protein, 'Cas').
wdef(caspase,protein,caspase).
wdef(caspase,protein,caspase).
wdef(cbl, protein, 'Cb1').
wdef(ccrsrh,protein,'CCRSrh').
wdef(cd28, protein, 'CD28').
wdef(cells, structure, cell).
wdef(cholesterol,smallmolecule,cholesterol).
```

```
wdef(cpp32,protein,'CPP32').
wdef(crkl, protein, 'CrkL').
wdef(ctf,substance,'COOH-terminal fragment').
wdef(cytokine, smallmolecule, cytokine).
wdef(cytosol, structure, cytosol).
wdef(djnk,protein, 'DJNK').
wdef(djun, protein, 'DJun').
wdef(dynamitin,protein,dynamitin).
wdef(erk, protein, 'ERK').
wdef(eto,smallmolecule,'ETO').
wdef(etoposide,smallmolecule,etoposide).
wdef(fad,disease,'familial Alzheimer''''s disease').
wdef(fyn, protein, 'Fyn').
wdef(gdp, smallmolecule,'GDP').
wdef(gelsolin,protein,gelsolin).
wdef(gp120,protein,'gp120').
wdef(grb2, protein, 'Grb2').
wdef(gst, protein, 'glutathione S-transferase').
wdef(gtp, smallmolecule,'GTP').
wdef(hsp70,protein,'HSP70').
wdef(human, species, human).
wdef(ikk, protein, 'IKK').
wdef(inactivated, state, inactive).
wdef(inactive,state, inactive).
wdef(jnk, protein, 'JNK').
wdef(jnk, protein, 'JNK').
wdef(jnk2, protein,' JNK2').
wdef(kap3,protein,kap3).
wdef(kdakt, protein, 'KDAkt').
wdef(kinase,protein, kinase).
wdef(kinectin,protein,kinectin).
wdef(klc,protein,klc).
wdef(lamin,protein,lamin).
wdef(myosins,protein,myosins).
wdef(nmdar,protein, 'NMDAR').
wdef(nmdar2b, protein, 'NMDAR2B').
wdef(ntf,substance,'NH2-terminal fragment').
wdef(p70s6k, protein, p70s6k).
wdef(p78s6k, protein, p78s6k).
wdef(parp,protein, 'poly(ADP-ribose)polymerase').
wdef(pdk1, protein, 'PDK1').
wdef(peptides, protein, peptide).
wdef(pkb, protein, 'PKB').
```

```
wdef(pkc,protein, 'protein kinase C').
wdef(position, site, site).
wdef(positions,site, site).
wdef(protease,protein,protease).
wdef(ps1,protein,'presenilin 1').
wdef(ps2,protein,'presenilin 2').
wdef(rap1, protein, 'Rap1').
wdef(ras, protein, 'Ras').
wdef(receptors, substance, receptor).
wdef(rela, protein, 'RelA').
wdef(residues,substance,residue).
wdef(responsive, state, active).
wdef(s6, protein, 'S6').
wdef(selectively, constraint, selective).
wdef(ser112, site, 'Ser112').
wdef(ser136, site, 'Ser136').
wdef(ser32, smallmolecule, 'Ser32').
phrase(ps1, protein
wdef(ser36, smallmolecule, 'Ser36').
phrase(ps1, protein, [ps1,'-',ctf], 'ps1-ctf',r).
wdef(sh2,domain, 'SH2').
wdef(sh3,domain,'SH3').
wdef(shc, protein, 'Shc').
wdef(signalsome, complex,signalsome).
wdef(sites, site,site).
wdef(sos, protein, 'Sos').
wdef(staurosporine,smallmolecule,staurosporine).
wdef(sts,smallmolecule,'STS').
wdef(tcr, complex, 'T-cell receptor').
wdef(tetracycline, smallmolecule,tetracycline).
wdef(thr229,aminoacid, 'Thr229').
wdef(thr308,aminoacid,'Thr308').
wdef(thr389, aminoacid, 'Thr389').
wdef(threonine,aminoacid,threonine).
wdef(tyrosine, aminoacid, tyrosine).
wdef(unresponsive, state,inactive).
wdef(unstimulated, state, inactive).
wdef(zvad,smallmolecule,'zVAD').
```

```
% lexsyn.pat
% revised March 17, 2000
%                  SYNTACTIC LEXICON FOR ACTIONS
% Contains syntactic entries for action type words and phrases
%
% synp(+Word1,+Wordlist,+Syn)
% synp: Word1 is first word of phrase, Wordlist is list of words i
n phrase
% synp: Syn is syntactic categorey
%
% synw(+Word,+Syn) is same as synp except there is no wordlist
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

synp(account,[account,for],v).
synp(account,[account,for],vp).
synp(accounted, [accounted,for],ved).
synp(accounted, [accounted,for],ven).
synp(accounting, [accounting,for],ving).
synp(accounting,[accounting,for],n).
synp(accounts, [accounts,for],vp).
synp(add, [add, up],vp).
synp(add,[add, up],v).
synp(added,[added, up],ved).
synp(added,[added, up],ven).
synp(adding,[adding, up],n).
synp(adding,[adding, up],ving).
synp(adds, [adds, up],vp).
synp(am, [am,a,means,of, producing],vp).
synp(am, [am,due,to],vp).
synp(are, [are,a,means,of, producing],vp).
synp(are, [are,due,to],vp).
synp(as,[as,a,result,of],prep).
synp(attributable,[attributable,to],vp). % ?
synp(attributed,[attributed,to],ven).
synp(based,[based,on],ven).
synp(based,[based,upon],ven).
synp(be, [be,a,means,of, producing],v).
synp(be, [be,due,to],v).
synp(because,[because,of],prep).
synp(been, [been,a,means,of, producing],ven).
synp(been, [been,due,to],ven).
synp(being, [being,a,means,of, producing],n).
synp(being, [being,a,means,of, producing],ving).
```

Appendix B

```
synp(being, [being,due,to],n).
synp(being, [being,due,to],ving).
synp(caused, [caused,by],ved).
synp(caused, [caused,by],ven).
synp(convey,[convey,a, signal],v).
synp(convey,[convey,a, signal],vp).
synp(conveyed,[conveyed,a, signal],ved).
synp(conveyed,[conveyed,a, signal],ven).
synp(conveying,[conveying, a, signal],ving).
synp(conveying,[conveying,a, signal],n).
synp(conveys,[conveys,a, signal],vp).
synp(dissociate,[dissociate, from],vp).
synp(dissociate,[dissociate,from],v).
synp(dissociated,[dissociated,from],ved).
synp(dissociated,[dissociated,from],ven).
synp(dissociates,[dissociates, from],vp).
synp(dissociating,[dissociating,from],n).
synp(dissociating,[dissociating,from],ving).
synp(dissociation,[dissociation, from],n).
synp(down,[down,'-',regulate],v).
synp(down,[down,'-',regulate],vp).    %  A down-regulates B        A
 --> B
synp(down,[down,'-',regulated],ved).
synp(down,[down,'-',regulated],ven).
synp(down,[down,'-',regulates],vp).
synp(down,[down,'-',regulating],n).
synp(down,[down,'-',regulating],ving).
synp(down,[down,'-',regulation],n).
synp(due,[due,to,the,fact,that],adj).
synp(due,[due,to],adj). % ?
synp(form,[form, complex],v).
synp(form,[form, complex],vp).
synp(formation,[formation, of, complex],n).
synp(formed,[formed, complex],ved).
synp(formed,[formed, complex],ven).
synp(forming,[forming, complex],n).
synp(forming,[forming, complex],ving).
synp(forms,[forms, complex],vp).
synp(had, [had,an,active,role,in],ved).
synp(had, [had,an,active,role,in],ven).
synp(has, [has,an,active,role,in],vp).
synp(have, [have,an,active,role,in],v).
synp(have, [have,an,active,role,in],vp).
```

```
synp(having, [having,an,active,role,in],n).
synp(having, [having,an,active,role,in],ving).
synp(is, [is,a,means,of, producing],vp).
synp(is, [is,due,to],vp).
synp(functions,[functions,as,a,negative,regulator,of],vp).
synp(function, [function,as,a,negative,regulator,of],vp).
synp(lead, [lead,to],v).
synp(leads, [leads,to],vp).
synp(leading, [leading,to],n).
synp(leading, [leading,to],ving ).
synp(leads, [leads,to],vp ).
synp(led,[led,to],ved).
synp(led,[led,to],ven).
synp(may,[may,be,responsible,for],vp).
synp(mediate,[mediate, a, signal], v).    %A mediates a signal to
B
synp(mediate,[mediate, a, signal], vp).
synp(mediated,[mediated, a, signal], ved).
synp(mediated,[mediated, a, signal], ven).
synp(mediates,[mediates, a, signal], vp).
synp(mediating,[mediating, a, signal], n).
synp(mediating,[mediating, a, signal], ving).
synp(mediation, [mediation,of, a, signal],n).
synp(n,[n,'-',acetylate],v).
synp(n,[n,'-',acetylate],vp).
synp(n,[n,'-',acetylated],ved).
synp(n,[n,'-',acetylated],ven).
synp(n,[n,'-',acetylates],vp).
synp(n,[n,'-',acetylating],n).
synp(n,[n,'-',acetylating],ving).
synp(n,[n,'-',acetylation],n).
synp(n,[n,'-',acylate],v).
synp(n,[n,'-',acylate],vp).
synp(n,[n,'-',acylated],ved).
synp(n,[n,'-',acylated],ven).
synp(n,[n,'-',acylates],vp).
synp(n,[n,'-',acylating],n).
synp(n,[n,'-',acylating],ving).
synp(n,[n,'-',acylation],n).
synp(n,[n,'-',glycosylate],v).
synp(n,[n,'-',glycosylate],vp).
synp(n,[n,'-',glycosylated],ved).
synp(n,[n,'-',glycosylated],ven).
```

```
synp(n,[n,'-',glycosylates],vp).
synp(n,[n,'-',glycosylating],n).
synp(n,[n,'-',glycosylating],ving).
synp(n,[n,'-',glycosylation],n).
synp(n,[n,'-',terminal,proteolysis],n).
synp(o,[o,'-',glycosylate],v).
synp(o,[o,'-',glycosylate],vp).
synp(o,[o,'-',glycosylated],ved).
synp(o,[o,'-',glycosylated],ven).
synp(o,[o,'-',glycosylates],vp).
synp(o,[o,'-',glycosylating],n).
synp(o,[o,'-',glycosylating],ving).
synp(o,[o,'-',glycosylation],n).
synp(only, [only,after],prep).
synp(prolyl, [prolyl,'-',4,'-',hydroxylate],v ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylate],vp).
synp(prolyl, [prolyl,'-',4,'-',hydroxylated],ved ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylated],ven ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylates],vp).
synp(prolyl, [prolyl,'-',4,'-',hydroxylating],n ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylating],ving ).
synp(prolyl, [prolyl,'-',4,'-',hydroxylation],n).
synp(result, [result,from],v).
synp(result, [result,from],vp).
synp(result,[result,in],v).
synp(result,[result,in],vp).
synp(resulted, [resulted,from],ved).
synp(resulted, [resulted,from],ven).
synp(resulted, [resulted,in],ved).
synp(resulted, [resulted,in],ven).
synp(resulting, [resulting,from],n).
synp(resulting, [resulting,from],ving).
synp(resulting, [resulting,in],n).
synp(resulting,[resulting,in],ving).
synp(results, [results,from],vp).
synp(results,[results,in],vp).
synp(set, [set, free],v).
synp(set, [set, free],v).
synp(set, [set, free],ved).
synp(set, [set, free],ved).
synp(set, [set, free],ven).
synp(set, [set, free],ven).
synp(set, [set, free],vp).
```

```
synp(set, [set, free],vp).
synp(sets, [sets, free],vp).
synp(sets, [sets, free],vp).
synp(setting, [setting, free],n).
synp(setting, [setting, free],n).
synp(setting, [setting, free],ving).
synp(setting, [setting, free],ving).
synp(suppress, [suppress, activity, of],v).
synp(suppress, [suppress, activity, of],vp).
synp(suppressed, [suppressed, activity, of],ved).
synp(suppressed, [suppressed, activity, of],ven).
synp(suppresses, [suppresses, activity, of],vp).
synp(suppressing, [suppressing, activity, of],n).
synp(suppressing, [suppressing, activity, of],ving).
synp(suppression, [suppression,of, activity, of],n).
synp(switch,[switch, on, the, activity, of],vp).
synp(switched,[switched, on, the, activity, of],ved).
synp(switched,[switched, on, the, activity, of],ved).
synp(switched,[switched, on, the, activity, of],ved).
synp(switched,[switched, on, the, activity, of],ved).
synp(switched,[switched, on, the, activity, of],ved).
synp(switches,[switches, on, the, activity, of],vp).
synp(up,[up,'-',regulate],v).  %  A up-regulates B B --> A
synp(up,[up,'-',regulate],vp).  %  A up-regulates B B --> A
synp(up,[up,'-',regulated], ved).
synp(up,[up,'-',regulated],ven).  %  A up-regulates B B --> A
synp(up,[up,'-',regulates], vp).
synp(up,[up,'-',regulating],n).  %  A up-regulates B B --> A
synp(up,[up,'-',regulating],ving).  %  A up-regulates B B --> A
synp(up,[up,'-',regulation],n).
synp(was, [was,a,means,of, producing],ved).
synp(was, [was,due,to],ved).
synp(were, [were,a,means,of, producing],ved). % ?
synp(were, [were,due,to],ved).
synw(acetylate,v).
synw(acetylate,vp).
synw(acetylated,ved).
synw(acetylated,ven).
synw(acetylates,vp).
synw(acetylating,n).
synw(acetylating,ving).
synw(acetylation,n).
synw(activate,v).
```

```
synw(activate,vp).
synw(activated,ved).
synw(activated,ven).
synw(activates,vp).
synw(activating,n).
synw(activating,ving).
synw(activation,n).
synw(add,v).
synw(add,vp).
synw(added,ved).
synw(added,ven).
synw(adding,n).
synw(adding,ving).
synw(addition,n).
synw(adds,vp).
synw(after,prep).
synw(aggregate ,v).
synw(aggregate ,vp).
synw(aggregated ,ved).
synw(aggregated ,ven).
synw(aggregates,vp).
synw(aggregating ,n).
synw(aggregating ,ving).
synw(aggregation ,n).
synw(arrest,n).
synw(arrest,v).
synw(arrest,vp).
synw(arrested,ved).
synw(arrested,ven).
synw(arresting,n).
synw(arresting,ving).
synw(arrests,vp).
synw(associate,v).
synw(associate,vp).
synw(associated,ved).
synw(associated,ven).
synw(associates,vp).
synw(associating,n).
synw(associating,ving).
synw(association,n).
synw(attach ,v).
synw(attach,vp).
synw(attached ,ved).
```

```
synw(attached ,ven).
synw(attaches,vp).
synw(attaching ,n).
synw(attaching ,ving).
synw(attachment,n).
synw(bind,v).
synw(bind,vp).
synw(binding,n).
synw(binding,ving).
synw(binds,vp).
synw(block,v).
synw(block,vp).
synw(blockage,n).
synw(blocked,ved).
synw(blocked,ven).
synw(blocking,n).
synw(blocking,ving).
synw(blocks,vp).
synw(bound,ved).
synw(bound,ven).
synw(break,v).
synw(break,vp).
synw(breakage, n).
synw(breaking,n).
synw(breaking,ving).
synw(breaks,vp).
synw(broke,ved).
synw(broken,ven).
synw(catalyzation,n).
synw(catalyze,v).
synw(catalyze,vp).
synw(catalyzed,ved).
synw(catalyzed,ven).
synw(catalyzes,vp).
synw(catalyzing,n).
synw(catalyzing,ving).
synw(causation,n).
synw(cause,n).
synw(cause,v).
synw(cause,ven).
synw(cause,vp).
synw(caused,ved).
synw(causes,vp).
```

```
synw(causing, n).
synw(causing, ving).
synw(cleavage,n).
synw(cleave,v).
synw(cleave,vp).
synw(cleaved,ved).
synw(cleaved,ven).
synw(cleaves,vp).
synw(cleaving,n).
synw(cleaving,ving).
synw(coimmunoprecipitate ,v).
synw(coimmunoprecipitate,vp).
synw(coimmunoprecipitated ,ved).
synw(coimmunoprecipitated ,ven).
synw(coimmunoprecipitates,vp).
synw(coimmunoprecipitating ,n).
synw(coimmunoprecipitating ,ving).
synw(coimmunoprecipitation ,n).
synw(combination ,n).
synw(combine ,v).
synw(combine ,vp).
synw(combined ,ved).
synw(combined ,ven).
synw(combines,vp).
synw(combining ,n).
synw(combining ,ving).
synw(conjugate ,v).
synw(conjugate ,vp).
synw(conjugated ,ve).
synw(conjugated ,ved).
synw(conjugates,vp).
synw(conjugating ,n).
synw(conjugating ,ving).
synw(conjugation ,n).
synw(connect ,vp).
synw(connect,v).
synw(connected ,ve).
synw(connected ,ved).
synw(connecting ,n).
synw(connecting ,ving).
synw(connection ,n).
synw(connects,vp).
synw(constrain,v).
```

```
synw(constrain,vp).
synw(constrained,ved).
synw(constrained,ven).
synw(constraining,n).
synw(constraining,ving).
synw(constrains,vp).
synw(constraint,n).
synw(coprecipitate,v).
synw(coprecipitate,vp).
synw(coprecipitated,ved).
synw(coprecipitated,ven).
synw(coprecipitates, vp).
synw(coprecipitating,n).
synw(coprecipitating,ving).
synw(coprecipitation ,n).
synw(copurification ,n).
synw(copurified ,ved).
synw(copurified ,ven).
synw(copurifies,vp).
synw(copurify ,vp).
synw(copurify,v).
synw(copurifying ,n).
synw(copurifying ,ving).
synw(couple ,vp).
synw(couple,v).
synw(coupled,ved).
synw(coupled,ven).
synw(couples,vp).
synw(coupling,n).
synw(coupling,ving).
synw(cut,n).
synw(cut,v).
synw(cut,ved).
synw(cut,ven).
synw(cut,vp).
synw(cuts,vp).
synw(cutting,n).
synw(cutting,ving).
synw(deactivate,v).
synw(deactivate,vp).
synw(deactivated,ved).
synw(deactivated,ven).
synw(deactivates,vp).
```

```
synw(deactivating,n).
synw(deactivating,ving).
synw(deactivation,n).
synw(death,n).
synw(demethylate,v).
synw(demethylate,vp).
synw(demethylated,ved).
synw(demethylated,ven).
synw(demethylates, vp).
synw(demethylating,n).
synw(demethylating,ving).
synw(demethylation, n).
synw(dephosphorylate, v).
synw(dephosphorylate, vp).
synw(dephosphorylated, ved).
synw(dephosphorylated, ven).
synw(dephosphorylates, vp).
synw(dephosphorylating, n).
synw(dephosphorylating, ving).
synw(dephosphorylation, n).
synw(die,v).
synw(die,vp).
synw(died,ved).
synw(died,ven).
synw(dies,vp).
synw(disassemble, v).
synw(disassemble, vp).
synw(disassembled, ved).
synw(disassembled, ven).
synw(disassembles, vp).
synw(disassembling, n).
synw(disassembling, ving).
synw(disassembly, n).
synw(discharge,n).
synw(discharge,v).
synw(discharge,vp).
synw(discharged,ved).
synw(discharged,ven).
synw(discharges,vp).
synw(discharging,n).
synw(discharging,ving).
synw(disengage,v).
synw(disengage,vp).
```

```
synw(disengaged,ved).
synw(disengaged,ven).
synw(disengagement,n).
synw(disengages,vp).
synw(disengaging,n).
synw(disengaging,ving).
synw(divide,v).
synw(divide,vp).
synw(divided,ved).
synw(divided,ven).
synw(divides,vp).
synw(dividing,n).
synw(dividing,ving).
synw(division,n).
synw(dying,n).
synw(dying,ving).
synw(enhance,v).
synw(enhance,vp).
synw(enhanced,ved).
synw(enhanced,ven).
synw(enhancement,n).
synw(enhances,vp).
synw(enhancing,n).
synw(enhancing,ving).
synw(express,v).
synw(express,vp).
synw(expressed,ved).
synw(expressed,ved).
synw(expressed,ven).
synw(expresses,vp).
synw(expressing,n).
synw(expressing,n).
synw(expressing,ving).
synw(expression,n).
synw(generate,v).
synw(generate,vp).
synw(generated,ved).
synw(generated,ven).
synw(generates,vp).
synw(generating,n).
synw(generating,ving).
synw(generation,n).
synw(hew,v).
```

```
synw(hew,vp).
synw(hewed,ved).
synw(hewed,ven).
synw(hewing,n).
synw(hewing,ving).
synw(hews,vp).
synw(hinder,v).
synw(hinder,vp).
synw(hindered,ved).
synw(hindered,ven).
synw(hindering,n).
synw(hindering,ving).
synw(hinders, vp).
synw(hindrance,n).
synw(inactivate,v).
synw(inactivate,vp).
synw(inactivated,ved).
synw(inactivated,ven).
synw(inactivates, vp).
synw(inactivating,n).
synw(inactivating,ving).
synw(inactivation, n).
synw(incite,v).
synw(incite,vp).
synw(incited,ved).
synw(incited,ven).
synw(incitement,n).
synw(incites,vp).
synw(inciting,n).
synw(inciting,ving).
synw(induce,v).
synw(induce,vp).
synw(induced,ved).
synw(induced,ven).
synw(induces,vp).
synw(inducing,n).
synw(inducing,ving).
synw(induction,n).
synw(influence,n).
synw(influence,v).
synw(influence,vp).
synw(influenced,ved).
synw(influenced,ven).
```

```
synw(influences,vp).
synw(influencing,n).
synw(influencing,ving). % ?
synw(inhibit,v).
synw(inhibit,vp).
synw(inhibited,ved).
synw(inhibited,ven).
synw(inhibiting,n).
synw(inhibiting,ving).
synw(inhibition,n).
synw(inhibits,vp).
synw(initiate,v).
synw(initiate,vp).
synw(initiated,ved).
synw(initiated,ven).
synw(initiates,vp).
synw(initiating,n).
synw(initiating,ving).
synw(initiation,vp).
synw(instigate,v).
synw(instigate,vp).
synw(instigated,ved).
synw(instigated,ven).
synw(instigates,vp).
synw(instigating,n).
synw(instigating,ving).
synw(instigation,n).
synw(interact,v).
synw(interact,vp).
synw(interacted,ved).
synw(interacted,ven).
synw(interacting,n).
synw(interacting,ving).
synw(interaction,n).
synw(interactions,n).
synw(interacts,vp).
synw(join ,vp).
synw(join,v).
synw(joined,ved).
synw(joined,ven).
synw(joining,n).
synw(joining,ving).
synw(joins,vp).
```

```
synw(juncture,n).
synw(liberate,v).
synw(liberate,vp).
synw(liberated,ved).
synw(liberated,ven).
synw(liberates,vp).
synw(liberating,n).
synw(liberating,ving).
synw(liberation,n).
synw(limit,v).
synw(limit,vp).
synw(limitation, n).
synw(limited,ved).
synw(limited,ven).
synw(limiting,n).
synw(limiting,ving).
synw(limits,vp).
synw(link,n).
synw(link,v).
synw(link,vp).
synw(linked,ved).
synw(linked,ven).
synw(linking,n).
synw(linking,ving).
synw(links, vp).
synw(mediate,v).
synw(mediate,vp).
synw(mediated,ved).
synw(mediated,ven).
synw(mediates,vp).
synw(mediating,n).
synw(mediating,ving).
synw(mediation,n).
synw(methylate, vp).
synw(methylate,v ).
synw(methylated,ved ).
synw(methylated,ven ).
synw(methylates, vp).
synw(methylating,n ).
synw(methylating,ving ).
synw(methylation, n).
synw(modification,n).
synw(modified,ved).
```

```
synw(modified,ven).
synw(modifies,vp).
synw(modify,v).
synw(modify,vp).
synw(modifying,n).
synw(modifying,ving).
synw(mutate,v).
synw(mutate,vp).
synw(mutated,ved).
synw(mutated,ven).
synw(mutates,vp).
synw(mutating,n).
synw(mutating,ving).
synw(mutation,n).
synw(overexpress,v).
synw(overexpress,vp).
synw(overexpressed,ved).
synw(overexpressed,ven).
synw(overexpresses,vp).
synw(overexpressing,n).
synw(overexpressing,ving).
synw(overexpression,n).
synw(pair,v).
synw(pair,vp).
synw(paired,ved).
synw(paired,ven).
synw(pairing,n).
synw(pairing,ving).
synw(pairs,vp).
synw(phosphorylate,n).
synw(phosphorylate,vp).
synw(phosphorylated,ved).
synw(phosphorylated,ven).
synw(phosphorylates,vp).
synw(phosphorylating,n).
synw(phosphorylating,ving).
synw(phosphorylation, n).
synw(promote,v).
synw(promote,vp).
synw(promoted,ved).
synw(promoted,ven).
synw(promotes,vp).
synw(promoting,n).
```

```
synw(promoting,ving).
synw(promotion,n).
synw(prompt,n).
synw(prompt,v).
synw(prompt,vp).
synw(prompted,ved).
synw(prompted,ven).
synw(prompting,n).
synw(prompting,ving).
synw(prompts,vp).
synw(react,v).
synw(react,vp).
synw(reacted,ved).
synw(reacted,ven).
synw(reacting,n).
synw(reacting,ving).
synw(reaction,n).
synw(reacts,vp).
synw(regulate,v).
synw(regulate,vp).
synw(regulated,ved).
synw(regulated,ven).
synw(regulates,vp).
synw(regulating,n).
synw(regulating,ving).
synw(regulation,n).
synw(release,n).
synw(release,v).
synw(release,vp).
synw(released,ved).
synw(released,ven).
synw(releases,vp).
synw(releasing,n).
synw(releasing,ving).
synw(removal,n).
synw(remove,v).
synw(remove,vp).
synw(removed,ved).
synw(removed,ven).
synw(removes,vp).
synw(removing,n).
synw(removing,ving).
synw(replace,v).
```

```
synw(replace,vp).
synw(replaced,ved).
synw(replaced,ven).
synw(replacement,n).
synw(replaces,vp).
synw(replacing,n).
synw(replacing,ving).
synw(repress, vp).
synw(repress,v).
synw(repressed,ved).
synw(repressed,ven).
synw(represses, vp).
synw(repressing,n).
synw(repressing,ving).
synw(repression,n).
synw(require,v).
synw(require,vp).
synw(required,ved).
synw(required,ven).
synw(requirement,n).
synw(requires,vp).
synw(requiring,n).
synw(requiring,ving).
synw(restrain, vp).
synw(restrain,v).
synw(restrained,ved).
synw(restrained,ven).
synw(restraining,n).
synw(restraining,ving).
synw(restrains, vp).
synw(restraint, n).
synw(sensitization, n).
synw(sensitize,  vp).
synw(sensitize, v).
synw(sensitized,ved).
synw(sensitized,ven).
synw(sensitizes,vp).
synw(sensitizing,n).
synw(sensitizing,ving).
synw(separate,v).
synw(separate,vp).
synw(separated,ved).
synw(separated,ven).
```

```
synw(separates, vp).
synw(separating,n).
synw(separating,ving).
synw(separation, n).
synw(sever,v).
synw(sever,vp).
synw(severance,n).
synw(severed,ved).
synw(severed,ven).
synw(severing,n).
synw(severing,ving).
synw(severs,vp).
synw(signal,v).
synw(signal,vp).
synw(signaled,ved).
synw(signaled,ved).
synw(signaled,ven).
synw(signaling,n).
synw(signaling,ving).
synw(signals,vp).
synw(split,n).
synw(split,v).
synw(split,ved).
synw(split,ven).
synw(split,vp).
synw(splits,vp).
synw(splitting,n).
synw(splitting,ving).
synw(stimulate,v).
synw(stimulate,vp).
synw(stimulated,ved).
synw(stimulated,ven).
synw(stimulates,vp).
synw(stimulating,n).
synw(stimulating,ving).
synw(stimulation, n).
synw(substitute,v).
synw(substitute,vp).
synw(substituted,ved).
synw(substituted,ven).
synw(substitutes,vp).
synw(substituting,n).
synw(substituting,ving).
```

```
synw(substitution,n).
synw(suppress, vp).
synw(suppress,v).
synw(suppressed,ved).
synw(suppressed,ven).
synw(suppresses, vp).
synw(suppressing,n).
synw(suppressing,ving).
synw(suppression,n ).
synw(tie,n).
synw(tie,v).
synw(tie,vp).
synw(tied,ved).
synw(tied,ven).
synw(ties,vp).
synw(transcribe,v).
synw(transcribe,vp).
synw(transcribed,ved).
synw(transcribed,ven).
synw(transcribes,vp).
synw(transcribing,n).
synw(transcribing,ving).
synw(transcription,n).
synw(tying,n).
synw(tying,ving).
synw(ubiquitinization,n).
synw(ubiquitinize,v).
synw(ubiquitinize,vp).
synw(ubiquitinized,ved).
synw(ubiquitinized,ven).
synw(ubiquitinizes,vp).
synw(ubiquitinizing,n).
synw(ubiquitinizing,ving).
synw(urge,n).
synw(urge,v).
synw(urge,vp).
synw(urged,ved).
synw(urged,ven).
synw(urges,vp).
synw(urging,n).
synw(urging,ving).
% the following are verbs connected with complexes
synw(form,v).
```

```
synw(form,vp).
synw(forms,vp).
synw(formed,ved).
synw(formed,ven).
synw(forming,n).
synw(formation,n).
synw(assemble,v).
synw(assemble,vp).
synw(assembles,vp).
synw(assembled,ved).
synw(assembled,ven).
synw(assembling,n).
synw(assembly,n).
synw(dissassemble,v).
synw(dissassemble,vp).
synw(dissassembles,vp).
synw(dissassembled,ved).
synw(dissassembled,ven).
synw(dissassembling,n).
synw(dissassembly,n).
synw(dissociate,v).
synw(dissociate,vp).
synw(dissociates,vp).
synw(dissociated,ved).
synw(dissociated,ven).
synw(dissociating,n).
synw(dissociation,n).
synw(recruit,v).
synw(recruit,vp).
synw(recruits,vp).
synw(recruited,ved).
synw(recruited,ven).
synw(recruiting,n).
synw(recruitment,n).
```

```
% lexsemact.pat
% revised March 17, 2000
%              SEMANTIC LEXICON OF ACTIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
% For genomics - the grammar tests for semantic and syntactic cate
gories
% separately for action type of categories; for substances the lex
ical
% entries are the same as in the medical area
% action type phrases have two entries: a semantic entry and a syn
tactic entry
% This lexicon contains the semantic entries for words and phrases

% semp is a lexical entry for phrasal lexicon
% semp(+Word1,+Sem,+Wordlist,+Targetform,+Features)
% semp specifies a semantic lexical definition for the genomics li
terature
% semp is equivalent to the predicate "phrase" in the medical area
% semp: Word1 is first word of phrase, Sem is semantic category
% semp: Wordlist is list of words in phrase, Targetform is output
form
% semp: Features is a list of 2 elements or the atom "def" represe
nting defaul
% semp: Features 1st element is rev or nrev meaning reversed or no
t reversed
% semp: Features 2nd element is a # specifying number of arguments
 for action
% semp: Features = def is equivalent to a list = [nrev,2]
% in case action has 1 argument, use [1,_]

%semw is a lexical entry for single word
% semw(+Word,+Sem,+Targetform,+Features)
% semw: the arguments are the same as for semp except there is no
Wordlist
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

:- multifile(semp/5).
:- multifile(semw/4).

semp(account,cause,[account,for],cause,[def]).
semp(accounted,cause,[accounted,for],cause,[def]).
```

Appendix C

```
semp(accounting,cause,[accounting,for],cause,[def]).
semp(accounts,cause,[accounts,for],cause,[def]).
semp(add,  attach, [add, up], attach, [def]).
semp(added,  attach, [added, up], attach, [def]).
semp(adds,  attach, [adds, up], attach, [def]).
semp(are, cause,[are,a,means,of, producing],cause,[def]).
semp(are,cause,[are,due,to],cause,[2,rev]).
semp(as,cause,[as,a,result,of],cause,[2,rev]).
semp(attributable,cause,[attributable,to],cause,[2,rev]).
semp(attributed,cause,[attributed,to],cause,[2,rev]).
semp(based,cause,[based,on],cause,[2,rev]).
semp(based,cause,[based,upon],cause,[2,rev]).
semp(because,cause,[because,of],cause,[2,rev]).
semp(convey, signal, [conveys,a, signal], signal, [def]).
semp(conveyed, signal, [conveyed,a, signal], signal, [def]).
semp(conveying, signal, [conveying, a, signal], signal, [def]).
semp(conveys, signal, [conveys,a, signal],signal, [def]).
semp(dissociate, release, [dissociate, from], release,[def]).
semp(dissociated, release, [dissociated,from], release,[def]).
semp(dissociates, release, [dissociates, from], release,[def]).
semp(dissociation, release, [dissociation, from], release,[def]).
semp(down,signal,[down,'-',regulate], signal, [def]).   %  A down-
regulates B       A --> B
semp(down,signal,[down,'-',regulated], signal, [def]).   %  A down
-regulates B       A --> B
semp(down,signal,[down,'-',regulates], signal, [def]).   %  A down
-regulates B        A --> B
semp(down,signal,[down,'-',regulation], signal, [def]).   %  A dow
n-regulates B      A --> B
semp(due,cause,[due,to,the,fact,that],cause,[2,rev]).
semp(due,cause,[due,to],cause,[2,rev]).
semp(form, attach, [form, complex], attach, [def]).
semp(formation, attach, [formation, of, complex], attach, [def]).
semp(formed, attach, [formed, complex], attach, [def]).
semp(forms, attach, [forms, complex], attach, [def]).
semp(had, cause,[had,an,active,role,in],cause,[def]).
semp(has, cause,[has,an,active,role,in],cause,[def]).
semp(have, cause,[have,an,active,role,in],cause,[def]).
semp(is, cause,[is,a,means,of, producing],cause,[def]).
semp(is,cause,[is,due,to],cause,[2,rev]).
semp(functions,inactivate,[functions,as,a,negative,regulator,of],i
nactivate,[def]).
semp(function,inactivate,[function,as,a,negative,regulator,of],ina
```

```
ctivate,[def]).
semp(lead, cause, [lead,to], cause,[def]).
semp(lead,cause1,[lead,to],cause,[def]).
semp(leading, cause, [leading,to], cause,[def]).
semp(leading,cause,[leading,to],cause,[def]).
semp(leads, cause, [leads,to], cause,[def]).
semp(leads,cause1,[leads,to],cause,[def]).
semp(led,cause,[led,to],cause,[def]).
semp(may, cause,[may,be,responsible,for],cause,[def]).
semp(mediate, signal, [mediate, a, signal], signal, [def]).    %A
mediates a signal to B
semp(mediated, signal, [mediated, a, signal], signal, [def]).    %
A mediates a signal to B
semp(mediates, signal, [mediates, a, signal], signal, [def]).    %
A mediates a signal to B
semp(mediation, signal, [mediation,of, a, signal], signal, [def]).
    %A mediates a signal to B
semp(n, createbond, [n,'-',acetylate],'N-acetylate',[def]).
semp(n, createbond, [n,'-',acetylated],'N-acetylate',[def]).
semp(n, createbond, [n,'-',acetylates],'N-acetylate',[def]).
semp(n, createbond, [n,'-',acetylation],'N-acetylate',[def]).
semp(n, createbond, [n,'-',acylate],'N-acylate',[def]).
semp(n, createbond, [n,'-',acylated],'N-acylate',[def]).
semp(n, createbond, [n,'-',acylates],'N-acylate',[def]).
semp(n, createbond, [n,'-',acylation],'N-acylate',[def]).
semp(n, createbond, [n,'-',glycosylate],'N-glycosylate',[def]).
semp(n, createbond, [n,'-',glycosylated],'N-glycosylate',[def]).
semp(n, createbond, [n,'-',glycosylates],'N-glycosylate',[def]).
semp(n, createbond, [n,'-',glycosylation],'N-glycosylate',[def]).
semp(n,breakbond,[n,'-',terminal,proteolysis],'n-terminal proteoly
sis',[def]).
semp(o, createbond, [o,'-',glycosylate], 'O-glycosylate',[def]).
semp(o, createbond, [o,'-',glycosylated], 'O-glycosylate',[def]).
semp(o, createbond, [o,'-',glycosylates], 'O-glycosylate',[def]).
semp(o, createbond, [o,'-',glycosylation], 'O-glycosylate',[def]).
semp(only,time,[only,after],'only after',[2,rev]).
semp(prolyl, createbond, [prolyl,'-',4,'-',hydroxylate],
                  'prolyl-4-hydroxylate',[def]).
semp(prolyl, createbond, [prolyl,'-',4,'-',hydroxylated],
                    'prolyl-4-hydroxylate',[def]).
semp(prolyl, createbond, [prolyl,'-',4,'-',hydroxylates],
                'prolyl-4-hydroxylate',[def]).
semp(prolyl, createbond, [prolyl,'-',4,'-',hydroxylation],
```

```
                    'prolyl-4-hydroxylate',[def]).
semp(result,cause,[result,from],cause,[2,rev]).
semp(result,cause,[result,in],cause,[def]).
semp(resulted,cause,[resulted,from],cause,[2,rev]).
semp(resulted,cause,[resulted,in],cause,[def]).
semp(resulting,cause,[resulting,from],cause,[2,rev]).
semp(resulting,cause,[resulting,in],cause,[def]).
semp(results,cause,[results,from],cause,[2,rev]).
semp(results,cause,[results,in],cause,[def]).
semp(set, release, [set, free], release ,[def]).
semp(set, release, [set, free], release ,[def]).
semp(sets, release, [sets, free], release ,[def]).
semp(setting, release, [setting, free], release ,[def]).
semp(suppress, inactivate, [suppress, activity, of], inactivate, [
def]).
semp(suppressed, inactivate, [suppressed, activity, of], inactivat
e, [def]).
semp(suppresses, inactivate, [suppresses, activity, of], inactivat
e, [def]).
semp(suppression, inactivate, [suppression,of, activity, of], inac
tivate, [def]).
semp(switch,  activate, [switch, on, the, activity, of],  activate
, [def]).
semp(switched,  activate, [switched, on, the, activity, of],  acti
vate, [def]).
semp(switches,  activate, [switches, on, the, activity, of],  acti
vate, [def]).
semp(up,signal,[up,'-',regulate], signal, [2,rev]).  % A up-regul
ates B B --> A
semp(up,signal,[up,'-',regulated], signal, [2,rev]).
semp(up,signal,[up,'-',regulates], signal,[2,rev]).
semp(up,signal,[up,'-',regulation], signal, [2,rev]).
semp(was, cause,[was,a,means,of, producing],cause,[def]).
semp(was,cause,[was,due,to],cause,[2,rev]).
semp(were, cause,[were,a,means,of, producing],cause,[def]).
semp(were,cause,[were,due,to],cause,[2,rev]).
semw(acetylate, createbond, acetylate,[def]).
semw(acetylated, createbond, acetylate,[def]).
semw(acetylates, createbond, acetylate,[def]).
semw(acetylation, createbond, acetylate,[def]).
semw(activate, activate, activate,[def]).
semw(activated, activate, activate,[def]).
semw(activates, activate, activate,[def]).
```

```
semw(activation, activate, activate,[def]).
semw(add, attach,  attach,[def]).
semw(added, attach,  attach,[def]).
semw(addition, attach,  attach,[def]).
semw(adds, attach,  attach,[def]).
semw(after,time,after,[2,rev]).    % temporal relations
semw(aggregate ,attach,attach,[def]).
semw(aggregated ,attach,attach,[def]).
semw(aggregates, attach, attach,[def]).
semw(aggregation ,attach,attach,[def]).
semw(arrest, inactivate, inactivate,[def]).
semw(arrested, inactivate, inactivate,[def]).
semw(arrests, inactivate, inactivate,[def]).
semw(associate,attach,attach,[def]).
semw(associated,attach,attach,[def]).
semw(associates, attach, attach,[def]).
semw(association,attach,attach,[def]).
semw(attach,attach,attach,[def]).
semw(attached ,attach,attach,[def]).
semw(attaches, attach, attach,[def]).
semw(attachment, attach, attach,[def]).
semw(bind,attach,attach,[def]).
semw(binding,attach, attach,[def]).
semw(binds,attach,attach,[def]).
semw(block,inactivate,inactivate,[def]).
semw(blocked,inactivate,inactivate,[def]).
semw(blocking,inactivate,inactivate,[def]).
semw(blocks,inactivate,inactivate,[def]).
semw(bound,attach, attach,[def]).
semw(break, breakbond,  'break bond',[def]).
semw(breakage, breakbond,  'break bond',[def]).
semw(breaks, breakbond,  'break bond',[def]).
semw(broke, breakbond,  'break bond',[def]).
semw(broken, breakbond,  'break bond',[def]). % case without break
bond
semw(catalyzation,promote,catalyze,[def]).
semw(catalyze,promote,catalyze,[def]).
semw(catalyzed,promote,catalyze,[def]).
semw(catalyzes,promote,catalyze,[def]).
semw(catalyzing,promote, catalyze,[def]).
semw(cause, cause,cause,[def]).
semw(caused, cause,cause,[def]).
semw(causes, cause,cause,[def]).
```

```
semw(cleavage, breakbond,  'break bond',[def]).
semw(cleave, breakbond,  'break bond',[def]).
semw(cleaved, breakbond,  'break bond',[def]).
semw(cleaves, breakbond,  'break bond',[def]).
semw(coimmunoprecipitate, attach, attach,[def]).
semw(coimmunoprecipitated ,attach,attach,[def]).
semw(coimmunoprecipitates, attach, attach,[def]).
semw(coimmunoprecipitation ,attach,attach,[def]).
semw(combination ,attach,attach,[def]).
semw(combine ,attach,attach,[def]).
semw(combined ,attach,attach,[def]).
semw(combines, attach, attach,[def]).
semw(conjugate ,attach,attach,[def]).
semw(conjugated ,attach,attach,[def]).
semw(conjugates,  attach, attach,[def]).
semw(conjugation ,attach,attach,[def]).
semw(connect ,attach,attach,[def]).
semw(connected ,attach,attach,[def]).
semw(connection ,attach,attach,[def]).
semw(connects,  attach, attach,[def]).
semw(constrain, inactivate, inactivate,[def]).
semw(constrained, inactivate, inactivate,[def]).
semw(constrains, inactivate, inactivate,[def]).
semw(constraint, inactivate, inactivate,[def]).
semw(coprecipitate,attach,attach,[def]).
semw(coprecipitated,attach,attach,[def]).
semw(coprecipitates,  attach, attach,[def]).
semw(coprecipitation ,attach,attach,[def]).
semw(copurification ,attach,attach,[def]).
semw(copurified ,attach,attach,[def]).
semw(copurifies,  attach, attach,[def]).
semw(copurify ,attach,attach,[def]).
semw(couple ,attach,attach,[def]).
semw(coupled,attach,attach,[def]).
semw(couples, attach, attach,[def]).
semw(cut, breakbond,  'break bond',[def]).   % leave breakbond onl
y?
semw(cuts, breakbond,  'break bond',[def]).
semw(deactivate, inactivate, inactivate,[def]).
semw(deactivated, inactivate, inactivate,[def]).
semw(deactivates, inactivate, inactivate,[def]).
semw(deactivation, inactivate, inactivate,[def]).
semw(death, process, death,[1]).
```

```
semw(demethylate, breakbond, demethylate,[def]).
semw(demethylated, breakbond, demethylate,[def]).
semw(demethylates, breakbond, demethylate,[def]).
semw(demethylation, breakbond, demethylate,[def]).
semw(dephosphorylate, breakbond,dephosphorylate,[def]).
semw(dephosphorylated, breakbond,dephosphorylate,[def]).
semw(dephosphorylates, breakbond,dephosphorylate,[def]).
semw(dephosphorylation, breakbond,dephosphorylate,[def]).
semw(die, process, death,[1]).
semw(died, process, death,[1]).
semw(dies, process, death,[1]).
semw(disassemble, release, release,[def]).
semw(disassembled, release, release,[def]).
semw(disassembles, release, release,[def]).
semw(disassembly, release, release,[def]).
semw(discharge, release, release,[def]).
semw(discharged, release, release,[def]).
semw(discharges, release, release,[def]).
semw(disengage, release, release,[def]).
semw(disengaged, release, release,[def]).
semw(disengagement, release, release,[def]).
semw(disengages, release, release,[def]).
semw(divide, breakbond,  'break bond',[def]).
semw(divided, breakbond,  'break bond',[def]).
semw(divides, breakbond,  'break bond',[def]).
semw(division, breakbond,  'break bond',[def]).
semw(dying, process, death,[1]).
semw(enhance,promote,promote,[def]).
semw(enhanced,promote,promote,[def]).
semw(enhancement,promote,promote,[def]).
semw(enhances,promote,promote,[def]).
semw(enhancing,promote,promote,[def]).
semw(express, generate,express,[def]). % can have either 1 or 2 ar
guments
semw(expressed, generate,express,[def]).
semw(expresses, generate,express,[def]).
semw(expressing, generate,express,[def]).
semw(expression,generate,express,[def]).
semw(generate,generate,generate,[def]).
semw(generated,generate,generate,[def]).
semw(generates,generate,generate,[def]).
semw(generating,generate,generate,[def]).
semw(generation,generate,generate,[def]).
```

```
semw(hew, breakbond,  'break bond',[def]).
semw(hewed, breakbond,  'break bond',[def]).
semw(hews, breakbond,  'break bond',[def]).
semw(hinder, inactivate, inactivate,[def]).
semw(hindered, inactivate, inactivate,[def]).
semw(hinders, inactivate, inactivate,[def]).
semw(hindrance, inactivate, inactivate,[def]).
semw(inactivate, inactivate, inactivate,[def]).
semw(inactivated, inactivate, inactivate,[def]).
semw(inactivates, inactivate, inactivate,[def]).
semw(inactivation, inactivate, inactivate,[def]).
semw(incite, activate, activate,[def]).
semw(incited, activate, activate,[def]).
semw(incitement, activate, activate,[def]).
semw(incites, activate, activate,[def]).
semw(induce, activate, activate,[def]).
semw(induced, activate, activate,[def]).
semw(induces, activate, activate,[def]).
semw(induction, activate, activate,[def]).
semw(influence, activate, activate,[def]).
semw(influenced, activate, activate,[def]).
semw(influences, activate, activate,[def]).
semw(influencing, activate, activate,[def]).
semw(inhibit, inactivate, inactivate,[def]).
semw(inhibited, inactivate, inactivate,[def]).
semw(inhibition, inactivate, inactivate,[def]).
semw(inhibits, inactivate, inactivate,[def]).
semw(initiate, activate, activate,[def]).
semw(initiated, activate, activate,[def]).
semw(initiates, activate, activate,[def]).
semw(initiattion, activate, activate,[def]).
semw(instigate, activate, activate,[def]).
semw(instigated, activate, activate,[def]).
semw(instigates, activate, activate,[def]).
semw(instigation, activate, activate,[def]).
semw(interact, interact, interact,[def]).
semw(interacted, interact, interact,[def]).
semw(interaction, interact, interact,[def]).
semw(interactions, interact, interact,[def]).
semw(interacts, react, interact,[def]).
semw(join ,attach,attach,[def]).
semw(joined ,attach, attach,[def]).
semw(joining, attach, attach,[def]).
```

```
semw(joins,  attach, attach,[def]).
semw(juncture, attach, attach,[def]).
semw(liberate, release, release,[def]).
semw(liberated, release, release,[def]).
semw(liberates, release, release,[def]).
semw(liberation, release, release,[def]).
semw(limit, inactivate, inactivate,[def]).
semw(limitation, inactivate, inactivate,[def]).
semw(limited, inactivate, inactivate,[def]).
semw(limits, inactivate, inactivate,[def]).
semw(link,attach,attach,[def]).
semw(linked,attach,attach,[def]).
semw(linking, attach,attach,[def]).
semw(links,attach, attach,[def]).
semw(mediate, promote, promote,[def]).
semw(mediated, promote, promote,[def]).
semw(mediates, promote, promote,[def]).
semw(mediation, promote, promote,[def]).
semw(methylate, createbond, methylate,[def]).
semw(methylated, createbond, methylate,[def]).
semw(methylates, createbond, methylate,[def]).
semw(methylation, createbond, methylate,[def]).
semw(modification,modify,modify,[def]).
semw(modified,modify,modify,[def]).
semw(modifies,modify,modify,[def]).
semw(modify,modify,modify,[def]).
semw(modifying,modify,modify,[def]).
semw(mutate,modify,mutate,[1]).
semw(mutated,modify,mutate,[1]).
semw(mutates,modify,mutate,[1]).
semw(mutating, modify,mutate,[1]).
semw(mutation,modify,mutate,[1]).
semw(overexpressed, generate,overexpress,[def]).
semw(overexpresses, generate,overexpress,[def]).
semw(overexpressing, generate,overexpress,[def]).
semw(overexpress, generate, express,[def]).
semw(overexpression,generate,overexpress,[def]).
semw(pair,attach,attach,[def]).
semw(paired,attach,attach,[def]).
semw(pairing,attach,  attach,[def]).
semw(pairs,attach,  attach,[def]).
semw(phosphorylate, createbond, phosphorylate,[def]).
semw(phosphorylated, createbond, phosphorylate,[def]).
```

```
semw(phosphorylates, createbond, phosphorylate,[def]).
semw(phosphorylation, createbond, phosphorylate,[def]).
semw(precede, cause, cause,[def]).
semw(preceded, cause, cause, [def]).
semw(precedes, cause, cause, [def]).
semw(preceding, cause, cause, [def]).
semw(promote,promote,promote,[def]).
semw(promoted,promote,promote,[def]).
semw(promotes,promote,promote,[def]).
semw(promotion,promote,promote,[def]).
semw(prompt, activate, activate,[def]).
semw(prompted, activate, activate,[def]).
semw(prompting, activate, activate,[def]).
semw(prompts, activate, activate,[def]).
semw(react, react, react,[def]).
semw(reacted, react, react,[def]).
semw(reaction, react, react,[def]).
semw(reactions, react, react,[def]).
semw(reacts, react, react,[def]).
semw(regulate, signal, signal,[def]).
semw(regulated, signal, signal,[def]).        % B is regulated by
A   A --> B
semw(regulates, signal, signal,[def]).
semw(regulation, signal, signal,[def]).
semw(release, release, release,[def]).
semw(released, release, release,[def]).
semw(releases, release, release,[def]).
semw(removal, breakbond, 'break bond ',[def]).
semw(remove, breakbond, 'break bond ',[def]).
semw(remove, breakbond, 'break bond ',[def]).
semw(removes, breakbond, 'break bond ',[def]).
semw(replace, substitute, substitute,[def]).
semw(replaced, substitute, substitute,[def]).
semw(replacement, substitute, substitute,[def]).
semw(replaces, substitute, substitute,[def]).
semw(repress, inactivate, inactivate,[def]).
semw(repressed, inactivate, inactivate,[def]).
semw(represses, inactivate, inactivate,[def]).
semw(repression, inactivate, inactivate,[def]).
semw(require, cause, cause,[2,rev]).
semw(required, cause, cause,[2,rev] ).
semw(requirement, cause, cause,[2,rev]).
semw(requires, cause, cause,[2,rev] ).
```

```
semw(requiring, cause, cause,[2,rev] ).
semw(restrain, inactivate, inactivate,[def]).
semw(restrained, inactivate, inactivate,[def]).
semw(restrains, inactivate, inactivate,[def]).
semw(restraint, inactivate, inactivate,[def]).
semw(sensitization, activate, activate,[def]).
semw(sensitize, activate, activate,[def]).
semw(sensitized, activate, activate,[def]).
semw(sensitizes, activate, activate,[def]).
semw(separate, breakbond,  'break bond',[def]).
semw(separated, breakbond,  'break bond',[def]).
semw(separates, breakbond,  'break bond',[def]).
semw(separation, breakbond,  'break bond',[def]).
semw(sever, breakbond,  'break bond',[def]).
semw(severance, breakbond,  'break bond',[def]).
semw(severed, breakbond,  'break bond',[def]).
semw(severs, breakbond,  'break bond',[def]).
semw(signal, signal, signal,[def]).
semw(signaled, signal, signal,[def]).
semw(signaling, signal, signal,[def]).
semw(signals, signal, signal,[def]).
semw(split, breakbond,  'break bond',[def]).
semw(splits, breakbond,  'break bond',[def]).
semw(splitting,breakbond, 'break bond',[def]).
semw(stimulate, activate, activate,[def]).
semw(stimulated, activate, activate,[def]).
semw(stimulates, activate, activate,[def]).
semw(stimulation, activate, activate,[def]).
semw(substitute,  substitute, substitute,[def]).
semw(substituted,  substitute, substitute,[def]).
semw(substitutes,  substitute, substitute,[def]).
semw(substitution,  substitute, substitute,[def]).
semw(suppress, inactivate, inactivate,[def]).
semw(suppressed, inactivate, inactivate,[def]).
semw(suppresses, inactivate, inactivate,[def]).
semw(suppression, inactivate, inactivate,[def]).
semw(tie,attach,attach,[def]).
semw(tied,attach,attach,[def]).
semw(ties,attach,attach,[def]).
semw(transcribe,generate,transcribe,[def]).
semw(transcribed,generate,transcribe,[def]).
semw(transcribes,generate,transcribe,[def]).
semw(transcribing,generate,transcribe,[def]).
```

```
semw(transcription,generate,transcribe,[def]).
semw(ubiquitinize, createbond, ubiquitinize,[def]).
semw(ubiquitinize, createbond, ubiquitinize,[def]).
semw(ubiquitinized, createbond, ubiquitinize,[def]).
semw(ubiquitinizes, createbond, ubiquitinize,[def]).
semw(urge, activate, activate,[def]).
semw(urge, activate, activate,[def]).
semw(urged, activate, activate,[def]).
semw(urges, activate, activate,[def]).
semw(urging, activate, activate,[def]).
semw(form,attach,attach,[def]).
semw(forms,attach,attach,[def]).
semw(formed,attach,attach,[def]).
semw(forming,attach,attach,[def]).
semw(formation,attach,attach,[def]).
semw(assemble,attach,attach,[def]).
semw(assembles,attach,attach,[def]).
semw(assembled,attach,attach,[def]).
semw(assembling,attach,attach,[def]).
semw(assembly,attach,attach,[def]).
semw(dissassemble,release,release,[def]).
semw(dissassembles,release,release,[def]).
semw(dissassembled,release,release,[def]).
semw(dissassembling,release,release,[def]).
semw(dissassembly,release,release,[def]).
semw(dissociate,release,release,[def]).
semw(dissociates,release,release,[def]).
semw(dissociated,release,release,[def]).
semw(dissociating,release,release,[def]).
semw(dissociation,release,release,[def]).
semw(recruit,attach,attach,[def]).
semw(recruits,attach,attach,[def]).
semw(recruited,attach,attach,[def]).
semw(recruiting,attach,attach,[def]).
semw(recruitment,attach,attach,[def]).
```

```
% edited Genome grammar - adapted from MedLEE's grammar for use with MedLEE
% this is to be used along with the genomics lexicon of substances, actions,
%   and relations.
% revised March 16, April 5, 2000
% adjusted for tagged input
:- multifile(wdef/3).
:- multifile(phrase/5).
%%%%%%%%%%%%%%%%%%%% Semantic Grammar for Genomics %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%     Written by Carol Friedman for the MedLEE System                          %
%
%     Queens College of the City University of New York                        %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Highest Level Predicate - sem_sent - 1st arg. is target structure            %
%                                    - 2nd arg. is a list of words in sentence%
%                                    - 3rd arg. is '[]'                        %
% Target structure: a frame or set of connected frames:                        %
%          the frame describes an action or several related actions;           %
%          an action frame is a list consisting of the symbol 'action'         %
%          followed by the code for the action and arguments.                  %
%          The arguments are either substances or actions;                     %
%          each substance slot consists of the name of the type of             %
%          substance followed by the value for the substance;                  %
%          the substance slot may contain slots for several substances.        %
% Examples:                                                                    %
% Blocking of il-2 gene transcription by activated rap1.                       %
% [action,inactivate,[protein,Rap1,[state,active]],                            %
%                    [action,transcribe,[x],[gene,interleukin-2]]]             %
%                                                                              %
% The adapter protein crkl was associated with both phosphorylated cbl and the%
% guanidine nucleotide-releasing factor c3g.                                   %
% [action,attach,[protein,CrkL],                                               %
%                [relation,and,[protein,Cb1,[state,phosphorylated]],           %
%                              [protein,guanidine nucleotide-releasing factor C3G,
%                                       [state,phosphorylated]]]] %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  fail an unknown predicate
:- unknown(_,fail).
:- op(900,  fy, [not,once]).  %  same priority and type as \+
:- op(700, xfx, [\=,~=]).     %  same priority and type as = or ==
% snoop is generally used to find input string when using a DCG
%       the input string is used for constraints
snoop(A,B,A,B).

sem_sent(P,Semlist,X) -->
        {assert(addstotal(0))},
        sem_parse(P,Semlist,X).
sem_parse(Target,Semlist) -->
        sem_patterns(P,Semlist).
sem_parse(Target,Semlist,X) -->
        sem_patterns(P,Semlist),
        sem_endornot(P,Target,X).

sem_parse([failure],_,X,_,_) :-
        addstotal(X).
sem_endornot(P,P,X)  -->   % P is target if there is an endmark
```

```
        sem_endmark,
        {addstotal(X)}.    % X is number of times reached endmark
sem_endornot(_,_,_,_,_)   :-  % did not reach endmark; update count and fail
        uptotal, fail.
sem_endornot(_,[failure],X,_,_) :-
        addstotal(X),  % X is number of times reached
        X >= 50.


% Finding patterns


sem_patterns(F,Semlist) -->
        pattern(F1,Semlist),
        {F1 \= []},         % 1st finding should not be empty
        morepattern(R,F2,Semlist), % connected patterns
        {getrelation(R,F1,F2,F)}.


/*************************************************************************
* The action pattern types are: pattern, nounactionpatt, actpatt, and    *
* nounactpatt.                                                           *
* pattern  --> actionarg(A1)                                             *
*              active or passive verb                                    *
*              actionarg(A2).                                            *
* pattern  --> nounactionpatt.                                           *
* pattern -->  actpatt.                                                  *
*************************************************************************/

% pattern is saved in a symbol table (st); check for success/failure 1st
% Case where pattern is in st and has been successful
pattern(Fmt,_) --> checkst(pattern,_,s,Fmt).
% Case where pattern is in st as a failure.
pattern(_,_) --> checkst(pattern,_,f,_), {!, fail}.

% pattern 5: an action pattern with a nominal verb
% Ps1 cleavage by zvad.
% apoptosis-induced cleavage of PS2 by zDEVD.
pattern(F,Semlist) -->
     snoop(S0,S0),
   { \+ checkst(pattern,5,_,_,S0,_),
     actionchk(Semlist) },
     nounactionpatt(F),
     snoop(S,S),
   { addst(pattern,5,s,F,S0,S)
  }.


% pattern 1: an action/substance acts on an action/substance
% the activation of rap1 inhibits the expression of il-2
% rap1 functions as a negative regulator of tcr-mediated il-2 gene
% transcription.
pattern(F,Semlist) -->    snoop(S0,S0), % S0 is the input string
   {  \+ checkst(pattern,1,_,_,S0,_),
     actionchk(Semlist),
     connectchk(Semlist)  },
     actionarg(A1),
```

```
      connectact(Sem,[v,vp,ved],Target,Features),
      actionarg(A2),
      snoop(S,S),  %ending sentence list
    { member(def, Features),
      modlist([A1,A2,Site],Mods);
      member(rev,Features),
      modlist([A2,A1,Site],Mods)),
      frame(F,action,Target,Mods),
      addst(pattern,1,s,F,S0,S)
    }.

% pattern 2: an action/substance was acted on by an action/substance
% The aggregation of bad was suppressed.
% The aggregation of bad was suppressed by the phosphorylation of jnk.
% Grb2 was associated with Cbl.
% Apoptosis-associated cleavage of endogenous PS1 was blocked by the
% treatment with zVAD.
pattern(F,Semlist) -->
      snoop(S0,S0), % S0 is the input string
     { \+ checkst(pattern,2,_,_,S0,_),
       actionchk(Semlist),
       connectchk(Semlist) },
       actionarg(A2),
       sem_beterm(_),    % was
       connectact(Sem,[ven],Target,Features), %activated
       optbyarg(A1),
       snoop(S,S),  %ending sentence list
    { (member(def, Features),
       modlist([A1,A2,Site],Mods);
       member(rev,Features),
       modlist([A2,A1,Site],Mods)),
       frame(F,action,Target,Mods),
       addst(pattern,2,s,F,S0,S)
    }.

% pattern 3: an action/substance acted on an action/substance
% bad induced phosphorylation of fyn.
% tcr and cd28-mediated il-2 transcription.
pattern(F,Semlist) -->
      snoop(S0,S0),
    { \+ checkst(pattern, 3,_,_,S0,_),
      actionchk(Semlist),
      connectchk(Semlist) },
      actionarg(A1),    % substance or basic action
    % optdash,
      connectacts(Sem,[vp,ven,ved],Target,Features),    % 'activated'
    % optof,
      actionarg(A2), % had pattern here
      snoop(S,S),
  { (member(def, Features),
      modlist([A1,A2,Site],Mods);
      member(rev,Features),
      modlist([A2,A1,Site],Mods)),
      frame(F,action,Target,Mods),
      addst(pattern,3,s,F,S0,S)
  }.
```

```
% pattern 4:  a simple action pattern with an active verb.
% Activated Raf-1 phosphorylates MEK-1.
pattern(F,Semlist) -->
     snoop(S0,S0),
     %check that sentence has an action word/phrase
   { \+ checkst(pattern, 4,_,_,S0,_),
     actionchk(Semlist) },
     actpatt(F),
     snoop(S,S),
   { addst(pattern,4,s,F,S0,S)
  }.

% no more patterns - save failure
pattern(_,_) -->  addst(pattern,0,f,_), {!, fail}.

%  sem_morepattern(-Rel,-P,+Semlist,+S0,+S):
%       Rel is a relation and its value frame;
%       P is the remaining patterns, Semlist is the list of semantic classes
%       in sentence
% if have a series of ','s, use the relation "and" or "or" if in the nest
% and make that the relation
morepattern(R,F,Semlist) -->
        sem_relation(R1,Mod1),   %relation and modifiers
        sem_patterns(F,Semlist),
        {( frame(F,rel,Conj2,_),  % F contains nested relation
            (Conj2 = and; Conj2 = or), frame(R1,rel,',',_), % R1 relation frame
           frame(R,rel,Conj2,_)   % value of relation is Conj2
           ;
           R1 \= [],  % where do Type, Value and Mods2 come from?
          frame(R1,Type,Value,Mod2),  % get components of original relation
          mergemods(Mod1,Mod2,Mods),
          ( Mods = [], frame(R,rel,Value,[]), !;
            %frame(R,rel,[Value|Mods],[])  % make it rel connector with rel mod
            R = [rel,[Value|Mods]]
          )
         )
        }.
% no more findings
morepattern([],[],_,S,S).

% actionarg is the argument of pattern
% actionarg is either a substance or a basic action

% actionarg is saved in a symbol table (st); check for success/failure 1st
% Case where actionarg is in st and have been successful
actionarg(A) --> checkst(actionarg,_,s,A).
% Case where actionarg is in st as a failure.
actionarg(_) --> checkst(actionarg,_,f,_), {!, fail}.

% actionarg 1: a substance or substances
% Rap1, active Rap1, Cbl and Crkl
actionarg(A) --> snoop(S0,S0), % S0 is the input string
              { \+ checkst(actionarg,1,_,_,S0,_)},
                substances(A),
                snoop(S,S),
              { addst(actionarg,1,s,A,S0,S) }.
```

```
% actionarg 2: a process like apoptosis, or a disease
actionarg(A) --> snoop(S0,S0), % S0 is the input string
                 { \+ checkst(actionarg,2,_,_,S0,_)},
                   processpatt(A),
                   snoop(S,S),
                 { addst(actionarg,2,s,A,S0,S)
   }.

% actionarg 3: a nominal action pattern
% Etoposide-induced apoptosis.
% Etoposide-induced PS1 cleavage by zVAD.
actionarg(A) --> snoop(S0,S0), % S0 is the input string
                 { \+ checkst(actionarg,3,_,_,S0,_)},
                   nounactionpatt(A),
                   snoop(S,S),
                   {addst(actionarg,3,s,A,S0,S)
    }.

% actionarg 4: the object of the nominal action is an actionarg
% Blocking of IL-2 Gene transcription by activated rap1.
actionarg(A) --> snoop(S0,S0), % S0 is the input string
                   { \+ checkst(actionarg,4 ,_,_,S0,_) },
                     action(Sem,[n,ving],Target,Features),
                     [of],
                     actionarg(A1),
                     optbyagent(A2),
                     snoop(S,S),
                   { (member(def, Features),
                     modlist([A1,A2],Mods);
                     member(rev,Features),
                     modlist([A2,A1],Mods)),
                     frame(A,action,Target,Mods),
                     addst(actionarg,4,s,A,S0,S)
     }.



% no more actionarg - save failure
actionarg(_) -->  addst(actionarg,0,f,_), {!, fail}.

% nounactionpatt is a nominal action pattern which allows for left and right
% modifiers
% Il-2 gene transcription mediated by tcr and cd28 was inhibited by rap1.
% Activated rap1 functions as a negative regulator of tcr and cd-28-mediated
il_2 transcription.
% nounactionpatt is saved in a symbol table (st); check for success/failure 1st
% Case where nounactionpatt is in st and has been successful
nounactionpatt(A) --> checkst(nounactionpatt,_,s,A).
% Case where nounaction patt is in st as a failure.
nounactionpatt(_) --> checkst(nounactionpatt,_,f,_), {!, fail}.

nounactionpatt(P) --> snoop(S0,S0),      % S0 is the input string
                      { \+ checkst(nounactionpatt,1 ,_,_,S0,_)},
                        actionlmod(L,Syn1),
                        nounactionunit(A),
                        actionrmod(R, Syn2),
```

```
                    snoop(S,S),
                  { (Syn1 = ved, append(R,[A], RA),
                     append(L, RA, P);
                     Syn1 = ving, append(R, [A], RA),
                     L = [action,Verb,Object],
                     modlist(RA, Object, Mods),
                     frame(P, action, Verb, Mods)),
                     addst(nounactionpatt,1,s,P,S0,S) }.
% no more nounactionpatt - save failure
nounactionpatt(_) -->  addst(nounactionpatt,0,f,_), {!, fail}.

% the central unit of the nounactionpatt is a nounactpatt or a process
nounactionunit(A) --> nounactpatt(A).
nounactionunit(A) --> process(A).

% left modifiers of nounactpatt
% Zvad-inhibited cleavage pf Ps1
actionlmod(L,ved) --> substances(S),
                      optdash,
                      action(Sem,[ved],Target,Features ),
                    { frame(L, action, Target, [S]) }.

% apoptosis induced cleavage of ps2
actionlmod(L,ved) --> process(S),
                      optdash,
                      action(Sem,[ved],Target,Features ),
                    { frame(L, action, Target, [S]) }.


% apoptosis causing cleavage of Ps1 by Zvad.
% need to invert the order of nounactpatt and actionlmod
actionlmod(L,ving) --> processobject(A),   % process or nounacpatt,
                       action(Sem,[ving],Target,Features),
                     { frame(L,action, Target,A) }.

actionlmod([],_) --> [].

actionrmod(R,ved) --> action(Sem,[ved],Target,Features),
                      byagent(A),  % may have to add ving to actionrmod
                    {  frame(R,action, Sem, A) }.
actionrmod([],_) --> [].


%
% actpatt parses a simple action between substances expressed by an active verb
%
% actpatt is saved in a symbol table (st); check for success/failure % % 1st
% Case where actpatt is in st and has been successful
actpatt(F) --> checkst(actpatt,_,s,F).
% Case where actpatt is in st as a failure.
actpatt(_) --> checkst(actpatt,_,f,_), {!, fail}.

% actpatt 1: substance acts on substance
% PDK1 phosphorylates p70s6k at Thr229
actpatt(F) -->
    snoop(S0,S0), % S0 is the input string
  { \+ checkst(actpatt,1 ,_,_,S0,_)},
```

```
    substances(A1),
    sem_whichrel,      % opt 'that'
    action(Semclass,[vp,ved],Target,Features),
    prepopt,  % added prepopt to allow action 'to' and 'with' substance
    substances(A2),
    siteinfo(Site),
    snoop(S,S),
  { (member(def, Features),
    modlist([A1,A2,Site],Mods);
    member(rev,Features),
    modlist([A2,A1,Site],Mods)),
    frame(F,action,Target,Mods),
    addst(actpatt,1 ,s,F,S0,S)
  }.

% acpatt 2:
% Substance was bound by Substance
% Substance was associated to substance.
% F can give either first or second place to the second argument;
% a byagent gets first position; prepagent gets second.
% Phosphorylated Fyn was associated with Cbl.

actpatt(F) -->
    snoop(S0,S0), % S0 is the input string
 {  \+ checkst(actpatt,2,_,_,S0,_)},
    substances(A1),
    sem_beterm(_),
    action(Semclass,[ven],Target,Features),
    optbyorprepagent(Position,A2),
    snoop(S,S),
 { (member(def, Features),
   (Position=second, modlist([A1,A2,Site],Mods);
    Position= first, modlist([A2,A1,Site],Mods));
    member(rev,Features),
   (Position=second, modlist([A2,A1,Site],Mods);
    Position= first, modlist([A1,A2,Site],Mods))),
    frame(F,action,Target,Mods),
    addst(actpatt,2,s,F,S0,S)
 }.

% no more actpatt - save failure
actpatt(_) -->  addst(actpatt,0,f,_), {!, fail}.

%
% nounactpatt parses a simple action between substances expressed by a nominal
% verb
%
% nounactpatt is saved in a symbol table (st); check for success/failure 1st
% Case where nounactpatt is in st and have been successful
nounactpatt(Fmt) --> checkst(nounactpatt,_,s,Fmt).
% Case where nounactpatt is in st as a failure.
nounactpatt(_) --> checkst(nounactpatt,_,f,_), {!, fail}.

% nounactpatt 1:
% Jnk phosphorylation of Bad
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
```

```
  { \+ checkst(nounactpatt,1,_,_,S0,_) },
    substances(A1),
    {aminoacidtest(A1)},
    optdash,
    action(Semclass,[n],Target,Features),
    ofobject(A2),
%   siteinfo(Site),
    snoop(S,S),
   {(member(def, Features),
     modlist([A1,A2,Site],Mods);
     member(rev,Features),
     modlist([A2,A1,Site],Mods)),
     frame(F,action,Target,Mods),
     addst(nounactpatt,1,s,F,S0,S)
   }.

% nounactpatt 2: the binding of substance and substance
% association of Fyn and Cbl.
% the reason for having this as a separate pattern is to
% prevent 'Fyn and Cbl' from being parsed together as substances
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
  { \+ checkst(nounactpatt,2 ,_,_,S0,_) },
    action(attach,[ving,n],Target,Features),
    ofobject1(A1),
    andobject(A2),
 %  siteinfo(Site),
    snoop(S,S),
  {  modlist([A1,A2,Site],Mods),
    frame(F,action,Target,Mods),
    addst(nounactpatt,2,s,F,S0,S)
   }.

% nounactpatt 3:
% The cleavage of protein by substance.
% Association of phosphorylated Fyn with Cbl
% Tyrosine phosphorylation of Cbl by kinase
% optbyorprepagent determines the order of arguments; byagent is placed first;
% prepagent is placed second

nounactpatt(F) -->
   snoop(S0,S0), % S0 is the input string
    { \+ checkst(nounactpatt,3 ,_,_,S0,_)},
    actionof(F),
    snoop(S,S),
  { addst(nounactpatt,3 ,s,F,S0,S) }.

actionof(F) -->
    siteinfo(Site),
    action(Semclass,[ving,n],Target,Features),
    optofobject(A1),
    optbyorprepagent(Position,A2),
    snoop(S,S),
  { (member(def, Features),
    (Position=second, modlist([A1,A2,Site],Mods);
     Position= first, modlist([A2,A1,Site],Mods));
     member(rev,Features),
```

```
    (Position=second, modlist([A2,A1,Site],Mods);
     Position= first, modlist([A1,A2,Site],Mods))),
     frame(F,action,Target,Mods)
  }.

% nounactpatt 4:
% Fyn association with Cbl.
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
  { \+ checkst(nounactpatt,4,_,_,S0,_) },
    substances(A1),
    action(Semclass,[ving,n],Target,Features),
    withobject(A2),
  % siteinfo(Site),
    snoop(S,S),
 {  modlist([A1,A2,Site],Mods),
    frame(F,action,Target,Mods),
    addst(nounactpatt,4,s,F,S0,S)
 }.

aminoacidtest(X) :-  X \= [aminoacid|_].

% nounactpatt 5:
% IL-2 gene transcription
% Cbl phosphorylation [by substance or action]
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
{   \+ checkst(nounactpatt,5 ,_,_,S0,_) },
    substances(A2),
    optdash,
    action(Semclass,[n],Target,Features),
    optbyagent(A1),
 %  siteinfo(Site),
    snoop(S,S),
 { (member(def, Features),
    modlist([A1,A2,Site],Mods);
    member(rev,Features),
    modlist([A2,A1,Site],Mods)),
    frame(F,action,Target,Mods),
    addst(nounactpatt,5 ,s,F,S0,S)
 }.

% nounactpatt 6:
% fyn-cbl association.
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
{   \+ checkst(nounactpatt,6 ,_,_,S0,_) },
    substances(A1),
    optdash,
    substances(A2),
    action(Semclass,[n,ving],Target,Features),
 %  siteinfo(Site),
    snoop(S,S),
  { modlist([A1,A2,Site],Mods),
    frame(F,action,Target,Mods),
    addst(nounactpatt,6,s,F,S0,S)
  }.
```

```
% nounactpatt 7:
% Cbl phosphorylated by fyn.
nounactpatt(F) -->
    snoop(S0,S0), % S0 is the input string
    { \+ checkst(nounactpatt,7 ,_,_,S0,_)},
    substances(A1),
    action(Semclass,[ven],Target,Features),
    [by],
    substances(A2),
 %  siteinfo(Site),
    snoop(S,S),
 %          {(member(def, Features),
    { modlist([A2,A1,Site],Mods),
 %           member(rev,Features),
 %           modlist([A1,A2,Site],Mods)),
      frame(F,action,Target,Mods),
      addst(nounactpatt,7,s,F,S0,S)
    }.

% no more nounactpatt - save failure
nounactpatt(_) -->  addst(nounactpatt,0,f,_), {!, fail}.



connectact(Sem,Syn,Target,Features) -->
      action(Sem,Syn,Target,Features),
     {member(Sem,[cause,cause1,activate,inactivate,signal,substitute,promote])}.

connectacts(Sem,Syn,Target,Features) -->
      connectact(Sem,Syn,Target,Features).

% aminoacid like tyrosine : ex.: tyrosine Cbl phosphorylation
% at position 201 Thr
siteinfo(S)  --> aminoacid(A),
                 {frame(S,site,[A],[])} .
siteinfo(S)  -->
                sitepreps,  % 'in', 'at'
                position(S).
siteinfo([]) --> [].
sitepreps    --> prepterm(in,_).
sitepreps    --> prepterm(at,_).
position(S)  --> [position],
                 sem_integerterm(I),
               { frame(S,site,I,[])}.



% The definitions of actions refer to the lexicons lexsynact.pl and lexsemact.pl
% Sem is the semantic class; Syn is the syntactic class
% F is the target
% oneaction was added for use with moreaction to allow parsing of conjoined
% actions

oneaction(activate,Syn,F,Features)     --> activateterm(Syn,F,Features),{!}.
oneaction(attach,Syn,F,Features)       --> attachterm(Syn,F,Features),{!}.
oneaction(breakbond,Syn,F,Features)    --> breakbondterm(Syn,F,Features),{!}.
```

```
oneaction(createbond,Syn,F,Features)  --> createbondterm(Syn,F,Features),{!}.
oneaction(inactivate,Syn,F,Features)  --> inactivateterm(Syn,F,Features),{!}.
oneaction(react,Syn,F,Features)       --> reactterm(Syn,F,Features),{!}.
oneaction(release, Syn,F,Features)     --> releaseterm(Syn,F,Features),{!}.
oneaction(signal,Syn,F,Features)      --> signalterm(Syn,F,Features),{!}.
oneaction(substitute,Syn,F,Features)  --> substituteterm(Syn,F,Features),{!}.
oneaction(transcribe,Syn,F,Features)  --> transcribeterm(Syn,F,Features),{!}.
oneaction(promote,Syn,F,Features)     --> promoteterm(Syn,F,Features),{!}.
oneaction(generate,Syn,F,Features)    --> generateterm(Syn,F,Features),{!}.
oneaction(cause,Syn,F,Features)    -->    causeterm(Syn,F,Features),{!}.

action(activate,Syn,F,Features)    --> activateterm(Syn,A1,Features),
                            moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(attach,Syn,F,Features)      --> attachterm(Syn,A1 ,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(breakbond,Syn,F,Features)   --> breakbondterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(createbond,Syn,F,Features)  --> createbondterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(inactivate,Syn,F,Features)  --> inactivateterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(react,Syn,F,Features)       --> reactterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(release, Syn,F,Features)     --> releaseterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(signal,Syn,F,Features)      --> signalterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(substitute,Syn,F,Features)  --> substituteterm(Syn,F,Features),
                           moreaction(Conj,Args),
                           {Conj = [],F =A1;
                           Conj\=[], mergemods([[action,A1]],Args,Actions),
                           frame(F1,relation, Conj,Actions), F = [F1]}.
action(transcribe,Syn,F,Features)  --> transcribeterm(Syn,F,Features),
```

```
                              moreaction(Conj,Args),
                              {Conj = [],F =A1;
                              Conj\=[], mergemods([[action,A1]],Args,Actions),
                              frame(F1,relation, Conj,Actions), F = [F1]}.
action(promote,Syn,F,Features)      --> promoteterm(Syn,F,Features),
                              moreaction(Conj,Args),
                              {Conj = [],F =A1;
                              Conj\=[], mergemods([[action,A1]],Args,Actions),
                              frame(F1,relation, Conj,Actions), F = [F1]}.
action(generate,Syn,F,Features)     --> generateterm(Syn,F,Features),
                              moreaction(Conj,Args),
                              {Conj = [],F =A1;
                              Conj\=[], mergemods([[action,A1]],Args,Actions),
                              frame(F1,relation, Conj,Actions), F = [F1]}.
action(cause,Syn,F,Features)     --> causeterm(Syn,F,Features),
                              moreaction(Conj,Args),
                              {Conj = [],F =A1;
                              Conj\=[], mergemods([[action,A1]],Args,Actions),
                              frame(F1,relation, Conj,Actions), F = [F1]}.

% binds, phosphorylates and activates
moreaction(Conj,Args) --> sem_conjrest(Conj1),
                              oneaction(Sem,Syn,A,Features),
                              moreaction(Conj2,Alist),
                             {Conj2 = [], Alist=[],Conj=Conj1, Args = [[action,A]];
                              Conj2 \= [], Conj = Conj2,
                              addmod([action,A],Alist,Args) }.
moreaction([],[],S,S).


passiveconnect(Sem,[ven],Target,Features) -->
                    sem_beterm(_),
                    connectact(Sem,[ven],Target,Features).


processpatt(A) --> disease(A).
processpatt(A) --> process(A).


optbyorprepagent(first,A)  --> byagent(A).
optbyorprepagent(second,A) --> prepagent(A).
optbyorprepagent(first,A)  --> [], {A = x}.

byorprepagent(first,A)  --> byagent(A).
byorprepagent(second,A) --> prepagent(A).

optbyagent(A) --> byagent(A).
optbyagent(A) --> [], {A = [x]}.

byagent(A) --> [by],
               substances(A).
byagent(A) --> [by],
               nounactionpatt(A).
prepagent(A) --> withobject(A).
prepagent(A) --> toobject(A).
% prepagent(A) --> andobject(A).
prepagent(A) --> ofobject(A).
```

```
% optprepagent(A) --> byagent(A).
optprepagent(A) --> ofobject(A).
optprepagent(A) --> withobject(A).
optprepagent(A) --> toobject(A).
optprepagent(A) --> andobject(A).
optprepagent(A) --> [], {A= [x]}.

ofobject(A) --> [of],
                nounactionpatt(A).
ofobject(A) --> [of],
                substances(A).
ofobject(A) --> [of],
                actionof(A).
ofobject1(A) --> [of],substance(A).  % to parse Binding of Fyn and Bad.
optofobject(A) --> ofobject(A).
optofobject([x]) --> [].

processobject(A) --> process(A). % can be expanded to nounactpatt, etc.

% optwithobject(A) --> withobject(A).
% optwithobject(A) --> [], {A = [x]}.
withobject(A) --> [with], substances(A).
toobject(A)   --> [to], substances(A).
andobject(A)  --> [and], substances(A).
prepobject(A) --> [to],substances(A).
prepobject(A) --> [with], substances(A).

optbyarg(A) --> [by],
                actionarg(A).
optbyarg(A) --> substances(A).
optbyarg(A) --> [], {A = ['substance unknown']}.

prepopt --> [to].
prepopt --> [with].
prepopt --> [by].
prepopt --> [of].
prepopt --> [].

% toopt
toopt --> [to].
toopt --> [].
% withopt
withopt --> [with].
withopt --> [].


optdash     --> ['-'].
optdash     --> [ ].
optof         --> [of].
optof         --> [ ].
/* optactionarg(A) --> actionarg(A).
optactionarg([]) --> []. */

optactionarg(A) -->
      actionarg(A).
```

```
% there is no further argument
optactionarg(A) -->
    [],
    {A = [] }.

% substances(F) --> substance(F).
% substances(F) --> substance(P1),
%             moresubstances(Conj,Plist),
%              { Conj = [], Plist = [], F = P1 ;
%              Conj \= [],
%            mergemods(P1,Plist,Args),
%               frame(F,relation,Conj,Args)
%              }.
% substances(F) --> substanceswithmods(F).
% substances(A) -->
%                 proteins(A).
% subswithmods.txt

% substances is saved in a symbol table (st);
% check for success/failure 1st
% Case where substances is in st and has been successful
substances(Fmt) --> checkst(substances,_,s,Fmt).
% Case where substance is in st as a failure.
substances(_) --> checkst(substances,_,f,_), {!, fail}.

substances(F) -->
        snoop(S0,S0),
      { \+ checkst(substances,1,s,_,S0,_)},
        lmods(Lmods),  % left modifiers
        (severalsubstances([relation,Conj,First|Rest]), % conjoined substances
        rmods(Rmods),     % right modifiers
% create list of lists containing distributed mods. of substances
      { distributesubs(Dist,[First|Rest],Lmods,Rmods),
% check Lmods - "no" F1 or F2 should be changed to no F1 and no F2
        fixconj(Lmods,[rel,Conj],[rel,C2]),
       %splice([Conj,Dist],F)
        frame(F,relation,C2,Dist)};
% substances and modifiers without conjunction
        substance(D1),
        rmods(Rmods),
        {D1 = [Type1, Substance1|ModsD1],
        delete(ModsD1, [], ModsD2),
        append([Lmods,Rmods],ModsD2,Allmods1),
        delete(Allmods1, [], Allmods2),
        frame(F,Type1,Substance1,Allmods2)}),
        snoop(S,S),
       {addst(substances,1,s,F,S0,S)}.

/* substances(F) --> snoop(S0,S0),
                {\+ checkst(substances,3 ,s,_,S0,_)},
                 complex(F),
                {addst(substances,3,s,F,S0,S)}.
*/
% no more substances- save failure
substances(_) -->  addst(substances,0,f,_), {!, fail}.
```

```
severalsubstances(F) --> substance(P1),
                         moresubstances(Conj,Plist),
                       {  Conj = [], Plist = [], F = P1 ;
                         Conj \= [],
                         addmod(P1,Plist,Args),
                         frame(F,relation,Conj,Args)
                        }.

%  ' X, Y, and Z'
 moresubstances(Conj,Args) --> sem_conjrest(Conj1),
                           substance(P1),
                           moresubstances(Conj2,Plist),
                        { Conj2 = [], Plist = [], Conj = Conj1, Args = [P1];
                           Conj2 \= [] ,Conj2\= /, Conj = Conj2,
                           addmod(P1,Plist,Args)
                         }.

% to allow for substances with modifiers
moresubstances(Conj1,Args) --> sem_conjrest(Conj1),
                               substances(Args),{!}.


moresubstances([],[]) --> [].  % no conjunction



% distributesubs
% distributes left mods and right mods over list of findings creating
% list of lists of findings with mods
distributesubs([],[],_,_) :- !.
distributesubs(Dist,[D1|Tail],Lmods,Rmods) :-
        distributesubs(Dist2,Tail,Lmods,Rmods), %distributed for remainder
        D1 = [Type1, Substance1|ModsD1],
        append([Lmods,Rmods],ModsD1,Allmods1),
        delete(Allmods1,[],Allmods2),
        frame(D,Type1,Substance1,Allmods2),
        append([D],Dist2,Dist). % Combine findings to get list of findings

lmods(A) --> stateterm(F),
             {frame(A, state, F, [])}.
lmods([]) --> sem_measure(_).
lmods([]) --> [].
rmods([]) --> [].
stateterm(F) --> acclex(state, F).
% for past participle of createbond and breakbond actions, the target
% is the word. ex.: phosphorylated, dephosphorylated, methylated
stateterm(F) -->
             snoop(S0,S0), % get the initial string
             createbondterm([ven], _,_),
             {S0 = [F|_]}. %get the first word of the string
stateterm(F) -->
             snoop(S0,S0), % get the initial string
             breakbondterm([ven], _,_),
             {S0 = [F|_]}. %get the first word of the string

% may have to add attachterm for 'bound'
```

```
% Taken from MedLEE grammar to handle '3 cm'
sem_measure(M) -->
                    sem_premeasure,
                    sem_quantityterm(N),
                    optdash,
                    sem_measureterm(Unit),
                  { frame(M,measure,[N,Unit],[]) }.
% complex predicates added November 8, 1999
% CrkL-C3G complex
% ras: raf-1 association
% ras: raf-1 complexes
% shc-grb2-sos
% TCR/CD3 complex
% p/CAF-p/CIP-CBP/p300-SRC-1 complex
% Ras:Raf-1 complexes
complex(C)  -->   proteins(P),
                  {P = [A,B|_],A \= [], B \= []},
                   optcomplexword,
                  { frame(C, complex,[P], []) }.

% a complex of NFAT4 with calcineurin
complex(C)   -->   complexword,
                   complexarg(A),
                   {frame(C, complex,[A],[])}.

complexarg(A) --> [of], proteins(A).
complexarg(A) --> [between], proteins(A).
% a complex between MyD88, IRAK-2, and the IL-1Rs
complexarg(A) --> action(contain), proteins(A).
% Complexes containing BOB.1/OBF.1 and Oct proteins

proteins(P)  --> protein(A),
                 moreproteins(P1),
                 {(A\=[]; append([A],P1,P))}.

moreproteins(A) --> proteinconnector,
                    proteins(A).

moreproteins([]) --> [].
proteinconnector  -->   ['-'].
proteinconnector  -->   ['/'].
proteinconnector  -->   [':'].
% connector  -->    [','].    taken out not to conflict with relation in
% connector  -->    [and].                                          moresubstances
proteinconnector(C)  --> [with].
optconnector -->    proteinconnector.
optconnector  -->   [].

complexword  -->  [complex].
complexword  -->  [complexes].
complexword  -->  ['signaling complexes'].

optcomplexword     --> complexword.
optcomplexword     --> [].


substance(A) --> protein(A).
```

```
substance(A) --> cell(A).
substance(A) --> species(A).
substance(A) --> structure(A).
substance(A) --> domain(A).
substance(A) --> gene(A).
substance(A) --> geneorprotein(A).
substance(A) --> aminoacid(A).
substance(A) --> smallmolecule(A).
substance(A) --> matter(A).
substance(A) --> proteinsite(A).
substance(A) --> disease(A).          % this will be modified later
substance(A) --> complex(A).

protein(A) -->
    proteinterm(P),
    {frame(A,protein,P,[])}.

complex(A) -->
    complexterm(P),
    {frame(A,complex,P,[])}.

cell(A) -->
    cellterm(P),
    {frame(A,cell,P,[])}.

species(A) -->
    speciesterm(P),
    {frame(A,species,P,[])}.

structure(A) -->
    structureterm(P),
    {frame(A,structure,P,[])}.

domain(A) -->
    domainterm(P),
    {frame(A,domain,P,[])}.

gene(A) -->
    geneterm(P),
    {frame(A,gene,P,[])}.

geneorprotein(A) -->
    gpterm(P),
    [X],
    {(X = gene, frame(A, gene, P, []);
      X = protein, frame(A, protein, P, []);
      X\= gene, X \= protein, frame(A, geneorprotein, P, []))}.

aminoacid(A) -->
    aminoacidterm(P),
    {frame(A,aminoacid,P,[])}.

smallmolecule(A) -->
    smallmoleculeterm(P),
    {frame(A,'small molecule',P,[])}.

matter(A) -->
```

```
    matterterm(P),
    {frame(A,substance,P,[])}.

proteinsite(A) -->
    proteinsiteterm(P),
    {frame(A,'protein site',P,[])}.

disease(A) -->
    diseaseterm(P),
    {frame(A,disease,P,[])}.
process(A) -->
     processterm(Syn,F,Features),
     {frame(A, process, F,[]),!}.
process(A) -->
     processterm(P),
     {frame(A, process, P, []),!}.


% terminals
proteinterm(F)         --> acclex(protein,F).
complexterm(F)         --> acclex(complex,F).
cellterm(F)            --> acclex(cell,F).
speciesterm(F)         --> acclex(species,F).
structureterm(F)       --> acclex(structure,F).
domainterm(F)          --> acclex(domain,F).
geneterm(F)            --> acclex(gene,F).
gpterm(F)              --> acclex(gp,F).
aminoacidterm(F)       --> acclex(aminoacid,F).
smallmoleculeterm(F)   --> acclex(smallmolecule,F).
matterterm(F)          --> acclex(substance,F).
proteinsiteterm(F)     --> acclex(proteinsite,F).
diseaseterm(F)         --> acclex(disease,F).
processterm(F)         --> acclex(process, F).

% action(activate,Syn,F,Features)    --> activateterm(Syn,F,Features).

activateterm(Syn,F,Features)  --> acclexss(activate, Syn,F,Features).
attachterm(Syn,F,Features)    --> acclexss(attach, Syn,F,Features).
breakbondterm(Syn,F,Features) --> acclexss(breakbond, Syn,F,Features).
createbondterm(Syn,F,Features)--> acclexss(createbond, Syn,F,Features).
inactivateterm(Syn,F,Features)--> acclexss(inactivate, Syn,F,Features).
reactterm(Syn,F,Features)     --> acclexss(react, Syn,F,Features).
releaseterm(Syn,F,Features)   --> acclexss(release, Syn,F,Features).
signalterm(Syn,F,Features)    --> acclexss(signal, Syn,F,Features).
substituteterm(Syn,F,Features)--> acclexss(substitute, Syn,F,Features).
transcribeterm(Syn,F,Features)--> acclexss(transcribe, Syn,F,Features).
promoteterm(Syn,F,Features)   -->  acclexss(promote, Syn,F,Features).
processterm(Syn,F,Features)   --> acclexss(process, Syn, F,Features).
generateterm(Syn,F,Features)  --> acclexss(generate,Syn,F,Features).
causeterm(Syn,F,Features)     --> acclexss(cause,Syn,F,Features).

% Semlist contains a phrase which is an action
actionchk(Semlist) :-
       intersect(Semlist,[attach,cause, createbond, breakbond,activate,
                 inactivate,substitute,transcribe,express,promote,signal]).

% Semlist contains a phrase which is a connector action
```

```
connectchk(Semlist) :-
       intersect(Semlist,[cause,activate,inactivate,substitute,
                          promote,signal]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Genome sectionc: ends here                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% relations are connected by conjunctions, or
%           certain 'conn' prepositions.
% Taken from MedLEE grammar to handle connectives that are conjunctions
%           Ex: "severe markings, possibly from tuberculosis"
sem_relation(F,[]) -->    % relation and modifiers
        sem_commapunc,
        sem_certainty([],C,rel),
        prepterm(P,conn),
        {frame(F,rel,P,C)}.
        %plice([[rel,P],C],R).

%           Ex: "markings, swelling",  "markings and swelling"
sem_relation(R,[]) -->  sem_conjrel(R),
                        sem_commapunc.
%          "density may represent known tumor"

%   "markings, and swelling"
sem_conjrel(F) -->
       sem_commapunc,
       sem_conjterm(Conj),
       {frame(F,rel,Conj,[])}.

sem_conjrest(Conj) -->     % restricted conj, has not sem_relation_showopt
        sem_commapunc,
        sem_conjterm(Conj).
% "markings, swelling"
sem_conjrest(',') -->
      snoop(S0,S0),
        sem_commapunc,
      snoop(S,S),
        {S0 \= S}.
% Treatment of Verbs from MedLEE's Grammar
%             form of "be"
sem_auxverb(B) --> sem_beterm(B).
%             form of "do"
sem_auxverb(B) --> sem_doterm(B).
%             form of "have"
sem_auxverb(B) --> sem_haveterm(B).

sem_recrel --> prepterm(in,_).
sem_recrel --> prepterm(to,_).
% "is not"
sem_auxrel(V) --> sem_auxverb(_),
                  sem_negterm(V).
sem_auxrel(V) --> sem_auxverb(V).
% left modifiers of findings include negation, quantity, certainty, degree, and
%                                    change type modifiers
```

```
sem_integer(W) --> [W],{integer(W)}.
sem_integer(W) --> integerterm(W).
sem_timeunit(T) --> sem_timeunitterm(T).

% From MedLEE grammar - "lasting 2 days", "for 2 days", "times 2 days"
sem_duration(F) -->
       sem_durpreps,
       sem_premeasure,  %about
       sem_timemeasure(T),
       sem_durationmod,  % opt. - "in duration"
       {frame(F,duration,[T],[])}.
sem_duration([],S,S).

sem_durpreps -->[times].
sem_durpreps -->
    prepterm(for,_).
sem_durpreps -->[lasting,for].
sem_durpreps -->[lasting].
sem_durpreps -->[lasted,for].
sem_durpreps -->[lasted].
sem_durationmod -->
         sem_aposts,  %opt. - "'s"
        [duration].
sem_durationmod --> [in], [duration].
sem_durationmod --> [].
sem_aposts --> [''''], [s].
sem_apost --> [].

% sem_frequency taken From MedLEE's grammar
% "two times", "times two", "two times a/per week", "two times daily"
sem_frequency(F) -->
        sem_freqterm(F1),    % "once"
        sem_freqterm(F2),     %  "a day"
        {frame(M,unitval,[F1,F2],[]),
         frame(F,frequency,[M],[])}.

sem_frequency(F) -->
        sem_freqterm(M),    % "qid", "daily"
        {frame(F,frequency,M,[])}.

% "2 times",
sem_frequency(F) -->
        sem_premeasure,
        sem_quantityterm(M),
        sem_times,
      {frame(F,frequency,[M],[])}.

% "times 2"
sem_frequency(Q) -->
        sem_times,
        sem_quantityterm(Q1),
        {frame(Q,frequency,Q1,[])}.
sem_frequency(F) -->
        [q], sem_quantityterm(Q),
             sem_timeunit(T),
        {frame(F,frequency,[unitval,[Q,T]],[])}.
```

```
sem_frequency(F) --> sem_eachevery,
                     sem_quantityterm(Q),
                     sem_timeunit(T),
                    {frame(F,frequency,[unitval,[Q,T,every]],[])}.
sem_frequency(Q) -->    % "second"
        sem_ordinal(O),
        sem_timeopt,
        {frame(Q,frequency,O,[])}.
sem_frequency([],S,S).
sem_timeopt  --> [time].
sem_timeopt  --> [].
sem_eachevery  --> [each].
sem_eachevery  --> [every].
sem_times-->[times].
sem_times-->[x].



% Taken from MedLEE's grammar
 negation modifier - "no" as in "no cardiomegaly"
sem_negation(F) -->
        sem_negterm(N),
        {frame(F,neg,N,[])}.
% negation not present
sem_negation([],S0,S0).

% Taken from MedLEE's grammar
% quantity modifier - "two" as in "two masses"
sem_quantity(F) -->
       snoop(S0,S0),
       { \+ checkst(sem_dates,1,s,_,S0,_) },  % not a legitimate date
       sem_quantityterm(Q),
       sem_quantityrmod(_),      % "2 or 3", "2 to 3"
       {\+ next_wordunit(S0),    % rule out '2 mm'
        frame(F,quantity,Q,[])
       }.
sem_quantity([],S0,S0).



sem_commapunc([','|S],S).
sem_commapunc(S,S).
sem_conjterm(C)     --> acclex(conj,C).
sem_doterm(D)       --> acclex(vdo,D).
sem_endmark([.|S],S).
sem_endmark([;|S],S).
sem_freqterm(F)     --> acclex(freq,F).
sem_haveterm(H)     --> acclex(vhave,H).
integerterm(I)      --> acclex(integer,I).
sem_measureterm(M)  --> acclex(unit,M).
sem_medterm(M)      --> acclex(med,M).
sem_negterm(N)      --> acclex(neg,N).
prepterm(P,C)       --> acclex(p,[P,C]).
sem_timeunitterm(T) --> acclex(timeunit,T).
```

```
% lexog - adapted from MedLEE lexicon
%%%%%%%%%%%%%%%%%%%  CLOSED WORD CATEGORY LEXICON %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%    NEGATIONS    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:-unknown(_,fail).
:-multifile(wdef/3).
wdef(cannot,neg,no).
wdef(neither,neg,no).
wdef(never,neg,no).
wdef(no,neg,no).
wdef(non,neg,no).
wdef(none,neg,no).
wdef(not,neg,no).
wdef(nothing,neg,no).
%%%%%%%%%%%%%%%%%%%    CONJUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef('&',conj,and).
wdef('/',conj,or).
wdef('-',grammar,'-').
wdef('+',conj,and).
wdef(although,conj,and).
wdef(and,conj,and).
wdef(as,conj,and).
wdef(because,conj,and).
wdef(but,conj,and).
wdef(',',conj,',').
wdef(except,conj,no).
%wdef(if,grammar,if).
wdef(minus,conj,no).
wdef(nor,conj,no).
wdef(or,conj,or).
wdef(that,grammar,that).
wdef(though,conj,and).
wdef(thru,conj,and).
wdef(verses,conj,or).
wdef(versus,conj,or).
wdef(vs,conj,or).
wdef(when,grammar,when).
wdef(where,grammar,where).
wdef(whereas,conj,and).
wdef(which,grammar,which).
wdef(while,conj,and).
wdef(who,grammar,who).
wdef(yet,conj,and).
%%%%%%%%%%%%%%%%%%%  PREPOSITIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef(above,ploc,above).
wdef(about,p,[approximately,nconn]).
wdef(about,ploc,about).
wdef(across,ploc,across).
wdef(abutting,ploc,near).
wdef(accompanies,p,[with,conn]).
wdef(accompanying,p,[with,conn]).
wdef(adjacent,ploc,adjacent).
wdef(adjacent,region,adjacent).
wdef(after,p,[after,conn]).
wdef(after,tprep,after).
wdef(along,p,[on,nconn]).
wdef(approximately,p,[approximately,nconn]).
wdef(around,p,[approximately,nconn]).
```

```
wdef(at,p,[at,nconn]).
wdef(atop,p,[on,nconn]).
wdef(before,ploc,before).
wdef(before,tprep,before).
wdef(behind,ploc,behind).
wdef(below,ploc,below).
wdef(between,ploc,between).
wdef(beyond,ploc,beyond).
wdef(by,ploc,near).
wdef(despite,p,[with,conn]).
wdef(during,p,[during,conn]).
wdef(during,tprep,during).
wdef(encasing,ploc,encasing).
wdef(extending,p,[in,nconn]).
wdef(following,p,[after,conn]).
wdef(following,tprep,after).
wdef(for,p,[for,nconn]).
wdef(from,p,[from,conn]).
wdef(in,p,[in,nconn]).
wdef(including,p,[with,conn]).
wdef(into,p,[in,nconn]).
wdef(involving,p,[of,nconn]).
wdef(next,tprep,next).
wdef(occupying,p,[in,nconn]).
wdef(on,p,[on,nconn]).
wdef(of,p,[of,nconn]).
wdef(over,ploc,over).
wdef(overlie,ploc,over).
wdef(overlied,ploc,over).
wdef(overlies,ploc,over).
wdef(overlying,ploc,over).
wdef(prior,tprep,before).
wdef(near,ploc,near).
wdef(radiating,ploc,radiating).
wdef(regarding,p,[about,nconn]).
wdef(roughly,grammar,roughly).      % 'roughly 6 mm'
wdef(since,p,[since,conn]).
wdef(since,status,subsequent).
wdef(through,p,[in,nconn]).
wdef(throughout,p,[in,nconn]).
wdef(to,p,[to,nconn]).
wdef(toward,p,[to,nconn]).
wdef(towards,p,[during,conn]).
wdef(under,ploc,below).
wdef(underneath,ploc,below).
wdef(until,tprep,until).
wdef(up,grammar,up).
wdef(upon,p,[on,nconn]).
wdef(via,p,[with,conn]).
wdef(with,p,[with,conn]).
wdef(within,p,[in,conn]).
wdef(without,p,[no,conn]).
%wdef(without,neg,no).

%%%%%%%%%%%%%%%%%%%%%%%%%% UNITS OF MEASURE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef('%',unit,percent).
```

```
wdef(cc,unit,cc).
wdef(centimeter,unit,cm).
wdef(centimeters,unit,cm).
wdef(cm,unit,cm).
wdef(degrees,unit,degree).
wdef(gm,unit,gram).
wdef(gms,unit,gram).
wdef(gram,unit,gram).
wdef(grams,unit,gram).
wdef(kg,unit,kilogram).
wdef(kilo,unit,kilogram).
wdef(kilogram,unit,kilogram).
wdef(kilograms,unit,kilograms).
wdef(liter,unit,liter).
wdef(liters,unit,liter).
wdef(microgram,unit,microgram).
wdef(micrograms,unit,microgram).
wdef(milliliter,unit,ml).
wdef(milliliters,unit,ml).
wdef(milligram,unit,mg).
wdef(milligrams,unit,mg).
wdef(milliseconds,unit,millisecond).
wdef(millivolts,unit,millivolt).
wdef(ml,unit,ml).
wdef(millimeter,unit,mm).
wdef(millimeters,unit,mm).
wdef(mm,unit,mm).
wdef(ozs,unit,ounce).
wdef(percent,unit,percent).
%%%%%%%%%%%%%%%%%%%%%%%%% NUMBERS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef(half,integer,'one half').
wdef(semi,quantity,semi).
wdef(ii,integer,2).
wdef(iii,integer,3).
wdef(vi,integer,4).
wdef(v,integer,5).
wdef(vi,integer,6).
wdef(vii,integer,7).
wdef(viii,integer,8).
wdef(ix,integer,9).
wdef(xii,integer,12).
wdef(xiii,integer,13).
wdef(one,integer,1).
wdef(two,integer,2).
wdef(double,quantity,double).
wdef(three,integer,3).
wdef(four,integer,4).
wdef(quadruple,quantity,quadruple).
wdef(five,integer,5).
wdef(six,integer,6).
wdef(sixty,integer,60).
wdef(seven,integer,7).
wdef(eight,integer,8).
wdef(nine,integer,9).
wdef(ten,integer,10).
wdef(eleven,integer,11).
wdef(twelve,integer,12).
```

```
wdef(thirteen,integer,13).
wdef(fourteen,integer,14).
wdef(fifteen,integer,15).
wdef(sixteen,integer,16).
wdef(seventeen,integer,17).
wdef(eighteen,integer,18).
wdef(nineteen,integer,19).
wdef(twenty,integer,20).
wdef(thirty,integer,30).
wdef(forty,integer,40).
wdef(fifty,integer,50).
wdef(sixty,integer,60).
wdef(seventy,integer,70).
wdef(eighty,integer,80).
wdef(ninety,integer,90).
wdef(hundred,integer,100).
wdef(thousand,integer,1000).
wdef(million,integer,1000000).
wdef(billion,integer,billion).
wdef(zero,integer,0).
wdef(first,ointeger,1).
wdef(second,ointeger,2).
wdef(third,ointeger,3).
wdef(fourth,ointeger,4).
wdef(fifth,ointeger,5).
wdef(sixth,ointeger,6).
wdef(seventh,ointeger,7).
wdef(eighth,ointeger,8).
wdef(ninth,ointeger,9).
wdef(tenth,ointeger,10).
wdef(eleventh,ointeger,11).
wdef(twelvth,ointeger,12).
wdef(thirteenth,ointeger,13).
wdef(fourteenth,ointeger,14).
wdef(fifteenth,ointeger,15).
wdef(sixteenth,ointeger,16).
wdef(seventeenth,ointeger,17).
wdef(eighteenth,ointeger,18).
wdef(ninteenth,ointeger,19).
wdef(triple,quantity,triple).
wdef(twentieth,ointeger,20).
wdef(thirtieth,ointeger,30).
wdef(single,quantity,1).
wdef(solitary,quantity,1).

wdef(frequency,grammar,frequency).*/
wdef('.',grammar,'.').
wdef(';',grammar,';').
wdef('/',grammar,'/').
wdef(':',grammar,':').
wdef('?',certainty,'moderate certainty').
wdef('+',certainty,'high certainty').
wdef('''',grammar,'''').

%%%%%%%%%%%%%%%%%%%%%%%%% FREQUENCIES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
wdef(once,freq,1).
wdef(times,grammar,x).
```

```
wdef(twice,freq,2).
```

```
% lexicon with lex0g containing common English words adapted from lex0 of
MedLEE%
% lex1g from lex1 of MedLEE
% August 23, 1999
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        CAROL FRIEDMAN                                      %
%          QUEENS COLLEGE, COLUMBIA UNIVERSITY                               %
%
%                       Version 3.0  4-01-00                                 %
%                       Version 2.0  1-31-96                                 %
%                       Version 1.0  1-5-92                                  %
%                                                                            %
%                                                                            %
%                       SEMANTIC LEXICON FOR CLINICAL TEXT                   %
%                                                                            %
%  The lexicon consists of several files:                                    %
%     lex0g.pl: single word closed classes                                    %
%     lex1g.pl: single word - general modifier type words:                    %
                                           %
%    wdef(category,target).                                                   %
%         word - is the name of the word being categorized;                   %
%         category - is the semantic category for the word                    %
%         target - is the canonical/standard form for the word                %
%                     words which are synonyms should be assigned the same    %
%                     canonical form.                                         %
%    multi-word phrases are categorized as follows:                           %
%     phrase(word,category,phrase,target).                                    %
%                                                                            %
%   Semantic Categories:                                                      %
%                                                                             %
%       certainty "possible"                                                  %
%               canonical values limited to: moderate -  for possible         %
%                                            high - for high possible         %
%                                            low - for low possible           %

%       conj - relational operators "and", "or" , which connect one finding %
%               to another finding                                            %
%       neg - negation "no", "not"
%       quant - for quantitative information "many"                           %

:-unknown(_,fail).
:-ensure_loaded([nsphrase,lex0g,lex1g,lexsemact,lexsyn,lexsub]).
```

```
% definitions kept from MedLEE lexicon - lex1.pl
wdef(be,vbe,'high certainty').
wdef(been,vbe,'high certainty').
wdef(being,vbe,'high certainty').
wdef(was,vbe,'high certainty').
wdef(is,vbe,'high certainty').
wdef(were,vbe,'high certainty').
/*
wdef(became,vcertainty,'high certainty').
wdef(become,vcertainty,'high certainty').
wdef(becomes,vcertainty,'high certainty').
wdef(becoming,vcertainty,'high certainty').
                          put in action lexicon
wdef(changed,change,change).
wdef(changes,change ,change).
wdef(changing,change,change).
wdef(necessarily,certainty,'high certainty').
wdef(necessary,vrecommend,recommended).
wdef(necessitate,vstatus,need).
wdef(necessitated,vstatus,need).
wdef(necessitating,vstatus,need).
wdef(necessitates,vstatus,need).
wdef(need,vstatus,need).
wdef(needed,vstatus,need).
wdef(needing,vstatus,need).
wdef(needs,vstatus,need).
                                                    */
```

```
% file ml_parser.pl
:- multifile(phrase/5).
:- multifile(wdef/3).
:-unknown(_,fail).
% Load in program components - library components are part of Prolog
:- ensure_loaded([library(basics),library(not),library(lists),
   library(readin),library(strings),library(ctypes),library(readconst),
   library(date), library(listparts), library(sets),
   radrec,radpardb,useful,util,tagging,lexicon, gengram]).

%:- initialization run.
%run :- on_exception(Error,processrun,stop(Error)).
runtime_entry(start) :- processrun.
runtime_entry(abort) :- halt.

% process report
processrun :- process, halt.

%stop(Error) :-
%   told,
%   write(user_error,'Error: '), write(user_error,Error), halt.
```

```
% get user supplied parameters and process report
process :-
get_args(Mode,Infile,Outfile,Prb,Undefs,Protocol), !,
           (Examtype = [];  % must have a domain
            process(Infile,Outfile,Prb,Undefs)).

% open Infile (text input) and process
process(Infile,Outfile,Prb,Undefs) :-
           see(Infile), seen, see(Infile),
           on_exception(Error,
           test_genome(Outfile,Prb,Undefs),
               app_err0(_,Outfile,Error)),
           closefiles(Outfile,Prb,Undefs).
process(_,Outfile,_,_) :-
        app_err(_,Outfile,'Program failed').

app_err0(_,Output,Error) :-
       tell(Output),
       write('<error>'),
       write('Prolog Error occurred: '),
       app_err(_,Output,Error).
app_err1(_,Output,Error) :-
       tell(Output),
       write('<error>'),
       write('Error in input: '),
       app_err(_,Output,Error).
app_err(_,Output,Error) :-
       tell(Output),
       write(Error), write('</error>'), nl.

closefiles(Outfile,Errfile,Unfile) :-
      tell(Outfile), told,
      (Errfile = []; tell(Errfile), told),
      (Unfile = []; tell(Unfile), told).
```

```
% Argument options - get user defined arguments
% -p ProbFile (otherwise default is problem messages are not written to file)
% -i Infile (if input is supplied by file and not standard input
% -s Section  (default is impression)
% -m Mode (default is relax; the three choices are strict, relax, skip)
% -o Outfile (if output should be file and not standard output)
% -? Provide list of default arguments
% -u Undefs (otherwise default is - undefined messages are not written
%     to a file)
get_args(Mode,Infile,Outfile,Prbfile,Undefs,Protocol) :-
    unix(args(Args)),
  (Args = [], !, writesyntax;
   Args = ['?'],!,  writesyntax;
   Args = [X|Rest], !,
   set_args([X|Rest],Mode,Infile,Outfile,Prbfile,Undefs,Protocol)).

writesyntax :-
     write(user_error,'geneparser [-m Mode]'),
     nl(user_error),
     write(user_error,'          [-t Outtype] [-p Probfile] [-u Undefs]'),
     nl(user_error),
     write(user_error,'          [-i Infile] [-o Outfile]'),
     nl(user_error).
```

```
% nsphrase.pl - contains words/phrases that are ignored
nosem(both,[both]).
nosem(however,[however]).
nosem(selectively,[selectively]).
nosem(specifically,[specifically]).
nosem(the,[the]).
nosem(a,[a]).
```

```
% file radpardb.pl
% June 25, 1999
%  fail an unknown predicate
 :-unknown(_,fail).
:- op(900,  fy, [not,once]).  %  same priority and type as \+
:- op(700, xfx, [\=,~=]).     %  same priority and type as = or ==
:- dynamic(sentno/1).
% \sem\radpardb.pl
%parse_sentences(+Beg,-Fmt,-ParseErrors,-Undefineds,-Unsents,+Section,
%                +UserMode,+Examtype,Sentno,Outsno,IncSno)
%       Beg is list of sentences, Fmt is list of target forms,
%       ParseErrors are a list of sentences which could not parse,
%       Undefineds is a list of undefined words in sentence
%       Unsents is a list of sentence containing undefined words
%       Section is the section of the examination, UserMode is the
%       parsing mode specified by user,
%       Examtype is the domain (type of exam)
%       Sentno is the number of the starting sentence
%       Outsno is the last sentence number + 1
%       IncSno is the amount that the sentence number should be increased
%            (i.e. it is 1 when called by parse_sects and 0 when in
%            recovery mode)
%    Each sentence is parsed independently.
parse_sentences([],[],[],[],[],_,_,_,_,_,_) :- !.  %no more sentences
parse_sentences(Beg,Fmtlist,Outfail,Outundefs,OutunSents,
                Section,UserMode,Examtype,_,_,IncSno) :-
    get_sentence(Beg,S,Rest),  !,
    ( isidentifier(S), !,   % ignore identifier sentences - parse remainder
      parse_sentences(Rest,Fmt1,Outfail,Outundefs,OutunSents,
                Section,UserMode,Examtype,_,_,IncSno), !,
       (outputform(htext), S \= ['.'], !, IncSno \= 0,  %0 means in recovery
mode
        append([[[sentence,S]]],Fmt1,Fmtlist);
        Fmtlist = Fmt1
       )
       ;
      %( IncSno = 0, !;    % on same sentence in recovery mode
      %  sentno(Sno), NewSentno is Sno + IncSno,
      %  retract(sentno(_)), assert(sentno(NewSentno))
      %%),

    % Incsno = 1, write('***'), write_list(S,3,_), nl, !,
     % Incsno = 0,

      preprocess(S,Bs,Undef,Semlist,strict),  % bracket and check for undefineds

      parse_modes(S,Bs,Semlist,Fmt1,Errors,Undef,Unsents,Section,Writefail,
                  Examtype,UserMode,IncSno), % parse first sentence

      parse_sentences(Rest,Fmt2,Moreerrors,Moreundefs,MoreUnSents,
                  Section,UserMode,Examtype,_,_,IncSno),  % parse remaining
      append(Errors,Moreerrors,Outfail),       % Combine failures
      (outputform(htext),
            (Fmt1 \= [], IncSno \= 0,
             !, append([Fmt1],Fmt2,Fmtlist);  % add extra bracket for 1st
             Fmt2 = [],  Fmtlist = Fmt1 , !
             )
```

```
        ;
        append(Fmt1,Fmt2,Fmtlist)
       ),              % Combine targets
       append(Unsents,MoreUnSents,OutunSents), % Combine sentences
       append(Undef,Moreundefs,Outundefs)      % Combine undefined words
      ).

%parse_modes(+S,+Bs,+Semlist,-Fmt,-Failures,+Undef,-Unsents,+Section,
%     +WriteMessage,+Examtype,+Mode,+IncSno)
%       S is original sentence; Bs is sentence after lexical lookup
%       Semlist is list of semantic categories in sentence
%       Fmt is formatted output,
%       Failures is list of sentences/fragments which could not be parsed.
%       Undef are words not in lexicon, Unsents are sentences containing
%               undefined words
%       Section is name of section being processed
%       WriteMessage is message returned from doresult (in case doresult fails)
%       Examtype is domain, Mode is user specified mode
%       IncSno is 0 if this is a fragment of a sentence that was already
%               parsed - but unsuccessfully; is 1 if this is a new sentence
% Best possible - try to get the most accurate parse possible trying
% all alternative strategies in turn if neccessary
% All words in sentence are defined
parse_modes(S,Bs,Semlist,Fmt,Errors,[],[],Section,no,Examtype,Pmode,
             Inc) :-
      (Pmode = bpseg, Pmodemod =  mode2, !; %in recovery mode
       Pmode = bpseg2, Pmodemod = mode2, !;
       Pmode = bpseg3, Pmodemod = mode2, !;
       Pmode = bpskip, Pmodemod = mode4, !; %in recovery mode
        % in user specified parse mode - don't parse in mode 5 or keyword
       Pmode \= keyword, Pmode \= mode5,
       Pmodemod = mode1
       ),
      dosent(S,Bs,Semlist,Fmt1,Message,Section,_,Examtype,Pmodemod,_),!, %
strict first
      recovery(_,S,Bs,Semlist,Fmt2,Message,Errors,[],[],Section,
                Pmode,Examtype,_), % try alternative modes if neccy
      (outputform(htext), Inc \= 0, !, append([[[sentence,S]],Fmt1,Fmt2],Fmt);
       append(Fmt1,Fmt2,Fmt)
      ).
% alternative strategies if have undefined words
parse_modes(S,Bs,Semlist,Fmt,Errors,Undef,Unsents,Section,no,Examtype,
             Pmode,Inc) :-
     Undef \= [],
     recovery(_,S,Bs,Semlist,Fmt1,yes,Errors,Undef,Unsents,Section,
                Pmode,Examtype,_), % try alternatives if have undefineds
     (outputform(htext), Inc\= 0, !, append([[sentence,S]],Fmt1,Fmt);
      Fmt = Fmt1
     ).
% key word strategy is fastest but least reliable;
parse_modes(S,Bs,Semlist,Fmt,Errors,Undef,Unsents,Section,no,Examtype,
             Pmode,Inc) :-
    (Pmode = keyword; Pmode = mode5
     ; Pmode = mode5),
     recovery(5,S,S,Semlist,Fmt1,yes,Errors,Undef,Unsents,Section,Pmode,
               Examtype,_),
     (outputform(htext), Inc \= 0, !, append([[sentence,S]],Fmt1,Fmt);
```

```
      Fmt1 = Fmt
     ).
% Parsing/Recovery modes
% parse_modes(+Level,+S,+Bs,+Sem,-Fmt,+Failed,+Undef,+Unsents,+Section,
%             +Pmode,+Examtype,_)
%   Level is the recovery level of the predicate
%   S is the original sentence list
%   Bs is the
%   Sem is the list of semantic categories in the sentence
%   Fmt is the formatted output for the sentence
%   Failed is 'yes' if the parse was unsuccessful, and 'no' otherwise
%   Undef is a list of words in sentence which are undefined(not in lexicon)
%   Unsents are the lists of sentences/segments which could not be parsed.
%   Section is the section of the report
%   Pmode is the user specified parse mode
%   Examtype is the domain
% mode 1 is the strictest parsing mode - the parser succeeded for the complete
%        original sentence using the grammar; all words in original sentence
%        are defined in lexicon
% mode 1 - alternative not needed because parse succeeded
recovery(1,_,_,_,[],no,[],Undef,Unsents,_,_,_,_) :- !.
%          ==no alternative strategy allowed in mode 1
%           in case where there are no undefineds, Noparse is S
recovery(1,S,_,_,[],yes,S,[],[],_,Pmode,_,_) :-
         Pmode = strict; Pmode = mode1, !.
%           in case there are undefineds, Unsents is S
recovery(1,S,_,_,[],yes,Noparse,Undef,Unsents,_,Pmode,_,_) :-
        (Pmode = strict; Pmode = 'mode1'),
        Undef \= [], Unsents = S, Noparse = [], !.
recovery(1,S,_,Semlist,[],yes,S,_,_,_,_,_,_) :-
% sentence contains no relev. information, don't try to recover
%     \+ (subtype(finding,Semlist); subtype(time,Semlist)), !.
\+ actionchk(Semlist).     % april 23, restored
% mode 4 - skip undefined words and try to parse according to mode 1
recovery(4,S,_,_,Fmt,yes,Errors,Undef,[],Sect,Pmode,Examtype,_) :-
         Undef \= [],
         (Pmode = bp; Pmode = mode4;
          Pmode = bpseg; Pmode = bpskip; Pmode = mode4
         ),
         preprocess(S,Bs,_,Semlist,bpskip),
         dosent(S,Bs,Semlist,Fmt1,Message,Sect,_,Examtype,mode4,_),!,
         recovery(_,Bs,Bs,Semlist,Fmt2,Message,Errors,[],[],Sect,
                     bpskip,Examtype,Sentno), % try alternatives if neccy
           append(Fmt1,Fmt2,Fmt).

% mode 3 - try longest parsed segment; partition rest of
%            sentence using mode 5 for parse mode bp
recovery(3,S,Bs,_,Fmt,yes,Errors,Undef,Unsents,Sect,Pmode,Examtype,_) :-
         % allowable modes for choosing longest segment
         (Pmode = bp; Pmode = bpskip;
          Pmode = skip; Pmode = mode3; Pmode = mode4;
          Pmode = bpseg3; Pmode = bpseg
         ),
         (Pmode = bpskip, Pmodemod = mode4_3;
          Pmodemod = mode3
         ),
         checkst(sem_pattern,_,s,Target,Bs,Rest),  %check symbol table
```

```
        %dooresult(Target,Fmt1,Examtype,Sect,Pmodemod,_),
         formatresult(Target,Pmodemod,Fmt1),
        (Pmode = mode3, Fmtlist = [], Errors = Rest;
        recovery(5,Rest,Rest,_,Fmtlist,yes,Errors,Undef,Unsents,Sect,
                        Pmode,Examtype,_)
        ),
        append(Fmt1,Fmtlist,Fmt).
% mode 2 segments sentence using word barrier methods. This mode is tried if
%          parse failed for original sentence/or there are undefined words
%            segment sentence using word barriers
recovery(2,S,_,_,Fmt,yes,Errors,Undef,Unsents,Sect,Pmode,Examtype,_) :-
        (Pmode = bp; Pmode = bpskip; Pmode = mode2; Pmode = skip;
         Pmode = mode2; Pmode = mode3; Pmode = mode4;
         Pmode = bpseg; Pmode = bpseg2;
         Pmode = bpseg3
         ),
        segmentandparse(S,Fmt,Errors,Unsents,Sect,Pmode,Examtype,_),!.
% mode 5 - try to partition sentences by findings
% when a finding in sentence is found, go left until first
%   modifier is found (if 2 findings are next to each other, 2nd one
%   is considered the finding and 1st is considered the modifier)
%   Repeat searching for successive findings using this method
recovery(5,[],[],_,[],_,[],_,_,_,_,_,_) :- !.
recovery(5,S,Bs,_,Fmt,yes,Errors,Undef,Unsents,Sect,
              Pmode,Examtype,_) :-
        (Pmode = bp; Pmode = bpskip; Pmode = bpseg; Pmode = keymode;
         Pmode = mode5; Pmode = negmode
        ),
         preprocess(S,Bs1,_,_,bpskip),  % skip undefined words
         actionfindingseg(Bs1,Fseg,Before),!,  % get segment containing finding
         (Fseg = [], Errors = S,  !; % no finding to segment
          %Before = [], Errors = Bs, Fmt1 = [], !;  % this part was tried
          preprocess(Fseg,Bseg,_,Semlist,bpskip),
          dosent(Fseg,Bseg,Semlist,Fmt1,Message,Sect,_,Examtype,
                  mode5,_)  % try to parse finding segment
         ),
          (Before = [], Before1 = [], Message = yes, !;  % no segmenting yet -
skip beg.
           Message = yes, Before1 = Before, !;  %don't add '.'; have to skip
more
           append(Before,['.'],Before1)
          ),
          ( Fseg = [], Fmt = [], !;  % no finding left in sent. - don't recover
          recoverrest(Fseg,_,Before1,Fmt2,Message,Errors,
                    Sect,Newmode,Examtype,_),
           % recover remainder
           append(Fmt1,Fmt2,Fmt)
          ).

% nothing could be recovered; all input -> Errors ; Format is []
recovery(_,Sents,_,_,[],yes,Sents,Undef,[],_,_,_,_).

%  part of phrase was skipped,  add period and treated skipped part as a
%  sentence
% recoverrest(+Segment,+Semlist,+Before,-Fmt,+Message,-Failures,+Section,
%       +Mode,+Examtype,_)
%       Segment is part of sentence with a finding
```

```
%       Semlist is a list of semantic categories for that sentence part
%       Before is the part of sentence before Segment
%       Fmt is the format for this segment
%       Message is 'no' if there is no segmantic information to be recovered
%               Message is 'yes' otherwise
%       Failures are lists of segment(s) that could not be parsed successfully
%       Section is section being processed, Mode is user specified parsing mode
%       Examtype is domain
recoverrest(_,_,Before,[],no,Before1,_,_,_,_) :-
  (Before = [], Before1 = [], !;   % nothing was skipped
   append(Before,['.'],Before1)
  ), !.
% nothing left to recover; write phrase that was skipped
recoverrest([],_,Before,[],yes,Before1,_,_,_,_) :-
   (Before = [], Before1 = [], !;
   append(Before,['.'],Before1)
   ), !.
% can recover partial parse
recoverrest(Bs,_,Before,Fmt,yes,Errors,Sect,Pmode,Examtype,_) :-
         checkst(sem_pattern,_,s,Target,Bs,Restseg), % recover from symbol tab.
         %doresult(Target,Fmt1,Examtype,Sect,mode5,_),
          formatresult(Target,mode5,Fmt1),
         recovery(5,Restseg,Rest,_,Fmt2,yes,Error2,
                  [],[],Sect,Pmode,Examtype,_),
         append(Fmt1,Fmt2,Fmt),
         (Before = [], Errors = Error2, !;   %nothing skipped to add '.' to
          append(Before,['.'|Error2],Errors)
         ).
% cannot recover partial parse - skip first element and retry
% if 1st element is a negation semantic type, skip 2nd element instead
%     Handles case where 1st element is a negation,certainty or status
%       add 2nd element to unparsed sentences list (enlcosed in angle brackets).
recoverrest([X,Y|Restseg],_,Before1,Fmt,yes,Errors,
                      Sect,Pmode,Examtype,_) :-
         foundword(X,Sem1,Tar),
         ( member(Sem1,[neg,certainty,vcertainty,vconn,status,vstatus]);
           Sem1 = p, Tar = [_,conn]
         ),
          %(Mod = neg; Mod = certainty; Mod = status; Mod = vcertainty), % leave
this mod in
          preprocess([X|Restseg],Fseg0,_,_,bpskip), % skip undefined words
          findingseg(Fseg0,Fseg,Before2), !,  % get finding seg
          (Fseg = [], Errors = [X,Y|Restseg], Fmt = []; % no finding
           preprocess(Fseg,Bseg,_,Restsem,bpskip), % skip undefined words
           dosent(Fseg,Bseg,Restsem,Fmt1,Message,Sect,_,Examtype,
                  mode5,_),  % try to parse finding segment
           recoverrest(Fseg,_,[Y|Before2],Fmt2,Message,Error2,
                     Sect,negmode,Examtype,_), % recover remainder
           (Before1 = [], Errors = Error2, !;
            append(Before1,[.|Error2],Errors)
           ),
           append(Fmt1,Fmt2,Fmt)
          ).
%   skip 1st element; enclose it in brackets
recoverrest([X|Restseg],_,Before1,Fmt,yes,Errors,
                Sect,Pmode,Examtype,_) :-
          preprocess(Restseg,Fseg0,_,_,bpskip),
```

```
            findingseg(Fseg0,Fseg,Before2), !,  % get finding seg
            append(Before1,[X|Before2],Before),
            (Fseg = [], Errors = [X|Restseg], Fmt = []; % no finding
             preprocess(Fseg,Bseg,_,Restsem,bpskip),
            dosent(Fseg,Bseg,Restsem,Fmt1,Message,Sect,_,Examtype,
                     mode5,_),  % try to parse finding segment
            recoverrest(Fseg,_,Before,Fmt2,Message,Errors,
                       Sect,Newmode,Examtype,_), % recover remainder
            append(Fmt1,Fmt2,Fmt)
            ).

%  no semantic information left; return Errors
recoverrest([X|Restseg],[],Before1,Fmt,yes,[X|Restseg],
                Sect,Pmode,Examtype,_).

%dosent(+S,+Bs,+Semlist,-Fmtlist,+Message,+Section,+WriteMessage,+Examtype,
%       +Mode)
%      S is original list of words in sentence; Bs is list after lexical lookup
%      Semlist is list of semantic categories corresponding to Bs
%      Fmtlist is list of target forms for sentence
%      Message is 'yes' if the output from parser signals a failure,
%                 and 'no' otherwise
%      Section is section of examination being processed
%      WriteMessage signals whether an error occurred in generating target form
%      Examtype is the domain, and Mode is the user specified mode of parsing
% Parse sentence and returns target in nested format
% Handles case where sentence should be skipped because info is about
%   family member or peripheral to patient
dosent(S,_,Semlist,[],Error,_,_,_,_,_) :-
  skipsentence(S,Semlist,Error), !.
dosent(S,Bs,Semlist,Fmtlist,Errormsg,Section,Writefail,Examtype,Mode,_) :-
   attemptparse(P,Bs,sentence,Semlist,Section,Atotal),
   (  P = [failure], Errormsg = yes, Writefail = no, ! % parse failure
      ;
      P = [], Errormsg = no, Writefail = no, Fmtlist = [], ! % empty target
      ;
      %doresult(P,Fmtlist,Examtype,Section,Mode,_),
        formatresult(P,Mode,Fmtlist),
        Errormsg = no, Writefail = no,!
      ;
      Errormsg = yes, Writefail = yes, !
   ).

%parse_sentences(Beg,Beg,[],[],_,_,_) :-  !.

% attemptparse(-P,+Bs,+Structure,+Semlist,-Ftype,-Total)
%       P is output from parser
%       Bs is list of words in sentence after lexical lookup
%       Structure is name of structure to be parsed
%       Semlist is list of semantic categories corresponding to elements in Bs
%       Total is number of times parser reached sem_sent in grammar;
%                where sem_sent is highest level predicate in grammar
% don't parse if sentence consists of only '.' or ';'
attemptparse([],Bs,_,_,_,_) :-
   Bs = ['.']; Bs = [';'].

% if a template exists for whole sentence, get parse from it
```

```
attemptparse(P,Bs,sentence,_,_,_) :-
   Bs = [X,'.'], is_list(X),  % the whole sentence is a finding
   find_sem_sent(P,X), !.

% parses and retracts wellformed string table - parses sentence
attemptparse(P,Bs,sentence,Semlist,Ftype,Atotal) :-
   retractall(wfst(_,_,_,_,_,_)),
   retractall(addstotal(_)),
   sem_sent(P,Semlist,Atotal,Bs,[]), !.

% parses and retracts wellformed string table - parses bodypart only
attemptparse(P,Bs,bodypart,_,_,_) :-
   sem_bodyloc(P,Bs,[]),
   retractall(wfst(_,_,_,_,_,_)), !.

%segmentandparse(+Sentences,-Fmtlist,-Failures,-Unsent,+Section,+Mode,
%       +Examtype,+Sentno)
%       Sentences is list of sentence segments.
%       Fmtlist consists of the formatted output for the segments
%       Failures is the list of unparsed segments.
%       Unsent is the list of segments with undefined words.
%       Section is the section being processed, Mode is the user specified mode
%       Examtype is the domain and Sentno is the sentence id.
segmentandparse([],[],[],[],_,_,_,_) :- !.
segmentandparse(Sentences,Fmtlist,Failures,UnSent,Section,Mode,
                  Examtype,Sentno) :-
     get_sentence(Sentences,S,Rest), !,  %sentence to segment
     preprocess(S,S1,_,Semlist,Mode), !,
     (Mode = mode2, NewPmode = bpseg2, !;
      Mode = mode3, NewPmode = bpseg3, !;
      NewPmode = bpseg
     ),
     (  segment1(S1,Segs,[],seg), !,
         parse_sentences(Segs,Fmt1,Fails,_,Un1,Section,NewPmode,Examtype,
                         Sentno,Sentno,0), !
      ; segment2(S1,Segs,[],seg), !,
         parse_sentences(Segs,Fmt1,Fails,_,Un1,Section,NewPmode,Examtype,
                         Sentno,Sentno,0), !
      ; segment3(S1,Segs,[],Negstatus,seg), !,
         parse_sentences(Segs,Fmt1,Fails,_,Un1,Section,NewPmode,Examtype,
                         Sentno,Sentno,0), !
     ),
       % fails if cannot segment sentence; otherwise segments remainder
      segmentandparse(Rest,Fmt2,Nexterrors,NextUns,Section,Mode,
                      Examtype,Sentno),
      append(Fmt1,Fmt2,Fmtlist),
      append(Un1,NextUns,UnSent),
      append(Fails,Nexterrors,Failures), !.

%segment1(+S,-Segs,+Beg,+Message)
%       S is list of words in sentence
%       Segs consists of sentence segments as separate sentences
%       Beg is list of words in sentence prior to the current portion of sentenc
%       Message is 'seg' if segmenting succeeded and 'noseg' otherwise
segment1([],[],_,noseg) :- !.
% segment sentence at connect phrase/word or at most conjunctions
% if negation precedes, restore negation
```

```
segment1([X|Rest],['.','<eos>'|Rem],Beg,seg) :-
     \+ sem_endmark(Rest,[]), % don't segment if at end already
     foundword(X,Sem,Target),   % get semantic classification and target
     ( X = nor, append([no],Rest,Rem)  % ok to segment at nor
      ;X = without, append([no],Rest,Rem)   % ok to segment at without
      %;X = ':', Rest = Rem
      ; Sem = neg, Rest = [Next|Rest2], % have negation; test word after
        foundword(Next,Sem2,Target2),   % for connective - add back negation
         testforconn(Next,Sem2,Target2), Rem = [X|Rest2]
      ; testforconn(X,Sem,Target), Rest = Rem
     ).

segment1([X|Rest],[X|Newrest],Start,Seg) :-
       append(Start,[X],Beg),    % part before segmentation
       segment1(Rest,Newrest,Beg,Seg).

testforconn(X,Sem,Target) :-
      ( Sem = p, Target = [P,conn],P\= with  % segment at connective prep
      ; member(Sem,[vconn,vshow])   % segment at these types of verbs
      ; Sem = conj, \+ member(X,[and,or,',','/',as])
      ).
% segment at certain words -
segment2([],[],[],noseg) :- !.

segment2(S,Segs,[],seg) :-
        seg2(S,Rest,Segs),
        \+ sem_endmark(Rest,[]), !.
segment2([X|Rest],[X|Newrest],[],Seg) :-
       segment2(Rest,Newrest,[],Seg).
seg2([X|Rest],Rest,['.','<eos>'|Rem]) :-
        member(X,[which,that,until,where,when,while,who,
         '(',')',between,whereby,after,before,prior,
         greater,ranging]),
        Rem = Rest, !.

segment3([],[],_,_,noseg) :- !.
% segment at conjunction - if negation preceded conjunction, add
segment3([X|Rest],Rem,Beg,Negstatus,seg) :-
       \+ sem_endmark(Rest,[]), !,  %  already at end of sentence
        seg3([X|Rest],Rem,Beg,Negstatus,seg), !.

seg3([X|Rest],Rem,Beg,Negstatus,seg) :-
        wdef(X,conj,_),
        member(X,[and,or,',']),
        (nonvar(Negstatus), Rem = ['.',Negstatus|Rest], ! %restore negation
         ; Rem = ['.','<eos>'|Rest], !
        ).
seg3([X|Rest],[X,'.','<eos>'|Rest],_,_,seg) :-
       foundword(X,age), !.

seg3([X|Rest],[X|Newrest],Start,Negstatus,Seg) :-
        ( nonvar(Negstatus), !;  % 1st neg already found  -  continue segmenting
        foundword(X,Sem,Target), !,
            ( Target = no, Negstatus = X, !;
              Sem = neg, Negstatus = X, !;
              Sem \= neg, Target \= no, !
            );
```

```
        true, !     % word is undefined
        ),
       append(Start,[X],Beg),     % part before segmentation
       segment3(Rest,Newrest,Beg,Negstatus,Seg), !.

% for finding type classes - parse as a sentence
whattoparse(Sem,P,Sent) :-

member(Sem,[cfinding,pfinding,morph,disease,device,proc,mproc,descriptor]),
        attemptparse(P,Sent,sentence,[Sem],impression,_).

% for bodyloc classes - parse as a bodyloc modifier
whattoparse(Sem,P,Sent) :-
        member(Sem,[bodyloc,region,side,position]),
        attemptparse(P,Sent,bodypart,_,_,_).
```

```
% file radrec.pl
% September 7, 1999
%  fail an unknown predicate
 :-unknown(_,fail).
:- op(900,  fy, [\+,not,once]).      %  same priority and type as \+
:- op(700, xfx, [\=,~=]).      %  same priority and type as = or ==
:- dynamic(domain/1).            %  domain being processed
:- dynamic(outputform/1).        % form of output (needed to distinguish
                                 % markup of text from formatting forms
:- dynamic(currentsect/1).       % section for outputting results

test_genome(Outfile,Errfile,Unfile) :-
     get_inputsents([],Toklist), !, % read in and tokenize input
     (Toklist = [], !,    % error condition
      app_err1(_,Outfile,'No input sent'), !
        ;
      parse_sentences(Toklist,Fmtlist,Failed,Undef,UnSent,impression,
bp,genome,_,_,0),!,
      outputresults(Fmtlist,Failed,Errfile,Undef,Unfile,UnSent,Outfile,
                    full,line,genome,1,0,_,exe,plain)
     ).

outputresults(Fmtlist0,Failed,Errfile,Undef,Unfile,UnSent,Outfile,
                Amount,Type,Exam,Compno,DocComp,NewCompno,Caller,Protocol) :-
      tell(Outfile),
     (Protocol = sgml, !, Op = sgml;
       Caller = server, !, Op = sgml;
        Op = plain),
      (Type = nested, !,  % original output form - nested findings
        write('<nested>'),new_line(Op),
         write(Fmtlist), new_line(Op), write('</nested>'),
         new_line(Op), !
       ),
     (Caller = server,
      write_message(Unfile,Undef,Caller,'<undefined>','</undefined>')
      ;
      Caller = exe, Undef \= [],
      write_message(Unfile,Undef,Caller,'***** Undefined Words *****',[])
      %write_highlight([],UnSent,Caller)
      ;
       true
      ),
     (Caller = server,
     write('<noparse>'),!,
     write_highlight(Undef,UnSent,Caller),
     write_highlight([],Failed,Caller), write('</noparse>')
     ;
     Caller = exe, Errfile \= [], Failed \= [],
     tell(Errfile),
     write('***** Sentences/Phrases Not Parsed *****'), nl,
     %write_highlight(Undef,UnSent,Caller),
     write_highlight([],Failed,Caller)
     ;
     true    % no Errfile to write to
    ).

% set_args: Process options
```

```
% Argument options
% -p ProbFile (otherwise default is problem messages are not written to file)
% -i Infile (if input is supplied by file and not standard input
% -m Mode (default is bp; the 6 choices are bp, mode1 - mode5)
% -o Outfile (if output should be file and not standard output)
% -? Provide list of default arguments
% -pr Protocol - sgml or plain (default is plain)
% -u Undefs (otherwise default is - undefined messages are not written
%     to a file)
set_args(Args,Mode,Infile,Outfile,Prbfile,Undef,Protocol) :-
      set_mode(Args,Mode), set_amount(Args,Amount),
      set_protocol(Args,Protocol),
      set_infile(Args,Infile), set_outfile(Args,Outfile),
      set_prbfile(Args,Prbfile), set_undefs(Args,Undef).

set_mode(Args,Mode) :-
    (nextto('-m',M,Args); nextto(m,M,Args)), !,
    modeis(M,Mode), !.
set_mode(_,bp).   % default output type

modeis(relax,mode2) :- !.
modeis(strict,mode1) :- !.
modeis(skip,mode4) :- !.
modeis(longest,mode3) :- !.
modeis(best,bp) :- !.
modeis(mode1,mode1) :- !.
modeis(mode2,mode2) :- !.
modeis(mode3,mode3) :- !.
modeis(mode4,mode4) :- !.
modeis(mode5,mode5) :- !.

set_protocol(Args,Protocol) :-
    (nextto('-pr',Protocol,Args); nextto('pr',Protocol,Args)),
     member(Protocol,[sgml,plain]), !.
set_protocol(_,plain).
set_undefs(Args,Undefs) :-
    nextto('-u',Undefs,Args); nextto(u,Undefs,Args) , !. % undef file option
set_undefs(_,[]).   % default is no file of undefineds created

set_infile(Args,Infile) :-
    nonvar(Infile), !;  % Infile is set already
    nextto('-i',Infile,Args), !;
    nextto(i,Infile,Args), !.
set_infile(_,user_input).   % default is standard input

set_prbfile(Args,Prbfile) :-
    nextto('-p',Prbfile,Args), !; nextto(p,Prbfile,Args), !. % prob file option
set_prbfile(_,[]).   % default is no file of problems is created

set_outfile(Args,Outfile) :-
    nonvar(Outfile), !;    % Outfile is already set
    nextto('-o',Outfile,Args), !; nextto(o,Outfile,Args), !. % outfile option
set_outfile(_,user_output).   % default is standard output

new_line(sgml) :-  write('<br>'), nl, !.
new_line(server) :- write('<br>'),nl, !.
new_line(exe) :- nl.
```

```
new_line(plain) :- nl.
write_message(_,[],exe,_,_) :- !.
write_message([],_,exe,_,_) :- !.
write_message(_,[],plain,_,_) :- !.
write_message([],_,plain,_,_) :- !.
write_message(File,Contents,Caller,Begmsg,Endmsg) :-
   ( member(Caller,[exe,plain]), tell(File), !
    ;
    true),
    write(Begmsg), new_line(Caller),
   (Contents = []; write_list(Contents,1), new_line(Caller)
   ),
   (Endmsg = [], !;
    write(Endmsg), !, new_line(Caller)
   ).

sentend([X|_],Caller) :-
   member(X,['.',';','?']), new_line(Caller), !.


gettargets([],[]) :- !.
gettargets([ignore|Rest],[ignore|Rest]) :-!.  % possibly ignore info.
gettargets([W1|Rest],[T1|Trest]) :-
     foundword(W1,_,T1),    % target for W1
     gettargets(Rest,Trest), !.
gettargets(W,W).  % not in lexicon
isneg(X) :-
    intersect(X,[no,negative,deny,'rule out']).

writeoutsent([Word|Rest]) :-
  write(''''), write(Word), write(''''), !,
  (Word = '''', write(''''), !; true),
  (Rest \= [], write(','), !, writeoutsent(Rest), !;
   true), !.
```

File
Tagging.pl

```
%  This file contains predicates associated with SGML tags
%  nextTag(+L,Tag,-PreTag,-PostTag) is true if
%     L is the starting List
%     Tag is an SGML tag; it could be a variable or instantiated already
%     PreTag is portion of L preceding Tag
%     PostTag is portion of L following Tag
nextTag(L,Tag,PreTag,PostTag) :-
    append(PreTag,['<',Tag,'>'|PostTag],L).

% endTag(+L,+Tag,-Pre,-Post) is true if
%     L is the starting list
%     Tag is the SGML end tag
%     Pre is the portion of L preceding the end of tag
%     Post is the portion of L following the end of tag
endTag(L,Tag,Pre,Post) :-
    append([Pre,['<','/',Tag,'>'],Post],L).

% enclosedPart(+L,+Tag,-Enclosed) is true if
%     L is the starting List; it is assumed that L is portion of some
%     list that follows a begin tag - i.e. '<',Tag|L
%     Tag is the SGML tag
%     Enclosed is the portion of text enclosed in tag; not including
%     end tag.
enclosedPart(L,Tag,Enclosed,Post) :-
    endTag(L,Tag,Enclosed,Post).
```

```
% file useful.pl - lexical lookup and utility tools
:-unknown(_,fail).
:-dynamic(sentence/1).
:- op(900,  fy, [not,once]).  %  same priority and type as \+
:- op(700, xfx, [\=,~=]).     %  same priority and type as = or ==
% useful.pl February 21, 1992
%
% preprocess(+S,+Bs1,-U,-Sem3,+Mode): preprocesses sentence to
%           bracket lexical phrases and remove words/phrases in
%           special db of noise words (nosem in nsphrase.pl db)
%       S is original sentence
%       Bs1 is preprocessed sentence
%       U is list of undefined words in sentence
%       Mode is mode of process - in skip mode undefined words are removed
%         from preprocessed sentence
preprocess(S0,Bs1,U,Sem3,Mode) :-      %cfnew
  checkbeg(S0,S),           % if beginning is 'A)' ignore
  checkphrase(S,S1,Sem1),  % bracket all phrases in phrasal lexicon first
  checklist(S1,U1,Bs,Sem2,Mode), % check that all words are in lexicon, remove
non semantic
  checklist(Bs,U,Bs1,Sem3,Mode). % check for phrases after non-sem are removed
 %append(Sem1,Sem2,Sem1),
 %append(Sem1,Sem3,Semlist),
  %union(U1,U2,U).
% found checks if word X is defined as a single word, or if X starts a defined
% phrase
foundword(X) :-
     wdef(X,_,_), !.
foundword(X) :-
      semw(X,_,_,_),!.
%definition from tagged input
foundword(X) :-
    phr(X,_,_,_), !.
foundword([X|Rest]) :-
       Rest \= [],
     phrasal(X,_,[X|Rest],_), !.
% 3/99 added foundword to search the new semact.pl lexicon
% phrasal using semp was added to util.lp
% found/2 returns semantic cat. of word
foundword(X,Sem) :-
     wdef(X,Sem,_).
foundword(X,Sem) :-
      semw(X,Sem,_,_).
%definition from tagged input
foundword(X,Sem) :-
      phr(X,Sem,[],_).
foundword([X|Rest],Sem) :-
     phrasal(X,Sem,[X|Rest],_).
% found/3 returns semantic cat. and target form
foundword(X,Sem,Form) :-
     wdef(X,Sem,Form).
foundword(X,Sem,Form) :-
      semw(X,Sem,Form,_).
%definition from tagged input
foundword(X,Sem,Form,_) :-
      phr(X,Sem,[],Form).
foundword([X|Rest],Sem,Form) :-
```

```
     phrasal(X,Sem,[X|Rest],Form).

%collectsem(+Word,-Sem): Sem is the list of semantic classes corresponding
%   to Word
collectsem(Word,Sem) :-
    setof(X,foundword(Word,X),Sem).
% missing checks if a word present in a sentence is defined
missing(X) :-
     member(X,S),
     not foundword(X).
% checkbeg(+S0,-S) checks beginning of sentence; if it begins with a letter or
% number followed by a ')', that part is skipped
checkbeg([X,')'|Rest],Rest) :- !.
checkbeg(X,X).

% checks every word in a list to see if it is defined; creates
% a new list of words not defined, and a new list of sentence
% where phrases are bracketed.
checklist([],[],[],[],_).
% if X is a list it has already been identified as a phrase in phrasal lex
checklist([X|Rest],Undef,Newrest,Semlist,Mode) :-
     is_list(X),
     check_no_sem([X|Rest],Rest1,_),
     checklist(Rest1,Undef,Newrest,Semlist,Mode), !.   %is phrase part of nosem
checklist([X|Rest],Undef,[X|Newrest],Semlist,Mode) :-
     %collectsem(X,Sem),
     is_list(X),   X = [W1|Tail],
     phrasal(W1,Sem,X,_),
     checklist(Rest,Undef,Newrest,Sem2,Mode) , !,
     append([Sem],Sem2,Semlist).
checklist([without|Rest],Undef,Newrest,Semlist,Mode) :-
     checklist([with,no|Rest],Undef,Newrest,Semlist,Mode).
% this problem has to be fixed in preprocessor
% check for a number with a ','  - "11,200"  and fix it
%checklist([X,',',Y|Rest],Undef,[N|Newrest],[number|Semlist],Mode) :-
%    number(X), number(Y), N is X * 1000 + Y, !,
%    checklist(Rest,Undef,Newrest,Semlist,Mode), !.
% check for a literal number    %cfnew
checklist([X|Rest],Undef,[X|Newrest],[number|Semlist],Mode) :-
     number(X) ,
     checklist(Rest,Undef,Newrest,Semlist,Mode), !.
% beginning of List is a prefix of a phrase that is a complete finding
checklist(List,Undef,[Phrase|Newrest],[cfinding|Semlist],Mode) :-
     check_sem_finding(List,Rest,Phrase),
     checklist(Rest,Undef,Newrest,Semlist,Mode) , !.
% beginning of List is a prefix of a phrase that is in nosemantic lexicon
checklist(List,Undef,Newrest,Semlist,Mode) :-
     check_no_sem(List,Rest,Phrase),
     checklist(Rest,Undef,Newrest,Semlist,Mode), !.
% beginning of List is a prefix of a phrase that is in phrasal lexicon
checklist(List,Undef,[Phrase|Newrest],Semlist,Mode) :-
     get_longest_sem(List,Rest,Phrase,Sem),
     %check_sem(List,Rest,Phrase,Sem),    %change to get longest phrase
     checklist(Rest,Undef,Newrest,Sem2,Mode), !,
     append(Sem,Sem2,Semlist).
% beginning of List is a single word that is in semantic lexicon
checklist([X|Rest], Undef,[X|Newrest],Semlist,Mode):-
```

```
      collectsem(X,Sem), !,
      %foundword(X,Sem), !,
      checklist(Rest, Undef,Newrest,Sem2,Mode), !,
      append(Sem,Sem2,Semlist).
% beginning of List is an undefined word
checklist([X|Rest],Undefs,Nrest,Semlist,Mode):-
      checklist(Rest,Undef,Newrest,Semlist,Mode),
      (member(X,Undef), !; Undefs = [X|Undef], !),
      (Mode = skip, !, Nrest = Newrest;
       Mode = bpskip, !, Nrest = Newrest;
       Nrest = [X|Newrest]), !.

% if beginning is a number followed by a . followed by a non number
% skip;   %cfnew
checkphrase([X,.],[X,.],[]) :-  !.
checkphrase([X,.,Z|Rest],Y,Semlist) :-
      number(X), not(number(Z)), checkphrase(Rest,Y,Semlist), !.
% beginning of List is a prefix of a phrase that is a complete finding
% or a phrase in phrasal lexicon
checkphrase(List,[Phrase|Newrest],Semlist) :-
      (check_sem_finding(List,Rest,Phrase), Sem = [cfinding];
       get_longest_sem(List,Rest,Phrase,Sem)
      ), !,
      %check_sem(List,Rest,Phrase,Sem)), !,
      checkphrase(Rest,Newrest,Sem2) , !,
      append(Sem,Sem2,Semlist).
checkphrase([W|Rest],[W|Newrest],Semlist) :-
      checkphrase(Rest,Newrest,Semlist).
checkphrase([],[],[]).

check_sem_finding([W|Tail],Tail,W) :-
            W = [W1|Rest],  % W is bracketed already
            sem_finding_sent(W1,W,_).
check_sem_finding([W|Tail],Sfinal,Phrase) :-
            sem_finding_sent(W,Phrase,_),
            begsublist(Phrase,[W|Tail],Sfinal), !.
sem_finding_sent(_,_,_) :- fail.
% check_no_sem(+Sent,-Rest,-Phrase): removes Phrase from Sent resulting
%    in Rest if Sent begins with a phrase in nosem (non-semantic list).
check_no_sem([W|Tail],Sfinal,Phrase) :-
            nosem(W,Phrase), %phrase beg. with W that should be removed
            begsublist(Phrase,[W|Tail],S1),
            remove_comma(S1,Sfinal), !.  % remove "," if it is next
%get_longest_sem(+Sent,-Rest,-Phrase,-Sem): Phrase is longest phrase that is
%  a prefix of Sent; Rest is remainder and Sem is list of semantic classes
get_longest_sem(Sent,Rest,Phrase,[Sem]) :-
        setof(X,check_sem(Sent,X),L),  % set of Phrases
        maxphrase(L,[],Phrase,0),      % Phrase with maximum length
        append(Phrase,Rest,Sent),      % rest of sentence after Phrase
        foundword(Phrase,Sem).

% check_sem(+Sent,-Rest,-Phrase,-Sem): checks if phrase beginning with
%       Sent is in phrasal lexicon; Rest is the remainder of Sent after phrase
%       Sem is the semantic class
check_sem([W|Tail],Rest,Phrase,Sem) :-
            phrasal(W,Sem,Phrase,_),
            begsublist(Phrase,[W|Tail],Rest).
```

```
%       this also obtains the Target form
check_sem([W|Tail],Rest,Phrase,Sem,Target) :-
           phrasal(W,Sem,Phrase,Target),
           begsublist(Phrase,[W|Tail],Rest).
check_sem([W|Tail],Tail,W,Sem) :-
           is_list(W),   %enclosed in brackets means it is a phrase
           W = [W1|Rest],
           phrasal(W1,Sem,W,_), !.
check_sem([W|Tail],Tail,W,Sem,Target) :-
           is_list(W),   %enclosed in brackets means it is a phrase
           W = [W1|Rest],
           phrasal(W1,Sem,W,Target), !.
% check_sem(+Sentence,-Phrase) is similar to check_sem/4 except for fewer args
check_sem(Sentence,Phrase) :-
          check_sem(Sentence,_,Phrase,_).
```

```
% file util.pl
%%%%%%%%%%%%%%%%  Utility Predicates  %%%%%%%%%%%

%  fail an unknown predicate
:-unknown(_,fail).
:- op(900,  fy, [not,once]).  %  same priority and type as \+
:- op(700, xfx, [\=,~=]).     %  same priority and type as = or ==

:- dynamic(wfst/6).
:- dynamic(addstotal/1).
:- dynamic(paragno/1).
:- dynamic(sectno/1).
:- dynamic(phr/4).

% wfst(+Rule,+Number,+Res,+Fmt,+S0,+S):  well-formed symbol table
%       Rule is the name of rule; Number is the option number
%       Res is s for success and f for failure
%       Fmt is the format (for successes); for failure Fmt is []
%       S0 is the sentence position at the start of Rule
%       S is the sentence position when Rule has been completed
%        add to wfst

addst(Rule,Number,Res,Fmt,S0,S) :-
    \+ checkst(Rule,Number,Res,Fmt,S0,S),  %result for rule was saved already
    \+ checkst(Rule,Number,i,Fmt,S0,S),    % result from different rule saved
   ( checkst(Rule,_,Res,Fmt,S0,S),   % different rule produced same result
      assert(wfst(Rule,Number,i,Fmt,S0,S));
    assert(wfst(Rule,Number,Res,Fmt,S0,S))), !.
addst(_,_,_,_,_,_):- !.    % always succeed

% checkst(+Rule,-Number,-Res,-Fmt,+S0,-S): checks to see if rule has been saved
%       in wfst
checkst(Rule,Number,Res,Fmt,S0,S) :-
    wfst(Rule,Number,Res,Fmt,S0,S).

% beglist(L,Y) - is Y the head of list L
beglist([X|_],Y) :-   X = Y , !.
% splice(+L1,-L2) : L1 is a list of lists; L2 is  merged list
splice(L1,L2) :- append(L1,L2), !.
%splice([],[]) :- !.
%splice([[]],[]) :- !.
%splice([X],X) :- !.
%splice([[]|L1],L2) :- splice(L1,L2),!.
%splice([[[]]|L1],L2) :- splice(L1,L2),!.
%splice([X|[[]]],L) :- splice(X,L),!.
%splice([L1,L2],L3) :-
%       append(L1,L2,L3), !.
%splice([X|L1],L2) :-
%        splice(L1,L3),
%        append(X,L3,L2) , !.

%splicerel - works with relations which have Arg1,...,Argn.
%            It splices a Splicelist in each arg of relation
splicerel(Finding,Splicelist,Spliced) :-
            splice(Splicelist,Sp1),
            (Finding = [rel,X|Rest], spliceargs(Rest,Sp1,Sp),
              %splice([[rel,X],Sp],Spliced),!;
```

```
                append([rel,X],Sp,Spliced),!;
                %splice([Finding,Sp1],Spliced) ).
                append(Finding,Sp1,Spliced) ).
%spliceargs - Splices a list into each element of a list
spliceargs([],_,[]) :-!.
spliceargs([Arg1|Rest],Splicelist,Spliced) :-
            %splice([Arg1,Splicelist],Sarg1),
            append(Arg1,Splicelist,Sarg1),
            spliceargs(Rest,Splicelist,Srest),
            %splice([[Sarg1],Srest],Spliced).
            append([Sarg1],Srest,Spliced).
list([],[]).
list([X|[]],X).
list([X|L1],L2) :- list(L1,L3),
                   append([X],L3,L2), !.

% strip(L1,L2) removes extra square brackets from L
strip([L],L).

% B is a suffix of A and C is the difference
difflist(A,B,C) :- append(C,B,A).
% S is a sublist at beg. of L if there is a list Rest, which when appended
%   to S results in L.
begsublist(S,L,Rest) :- append(S,Rest,L), !.
% checks that first element in list S has semantic category in Semlist
firstword([W1|_],Semlist) :-
    atom(W1), wdef(W1,Sem,_),  % semantic category
    member(Sem,Semlist).
firstword([W1|_],Semlist) :-
    is_list(W1), phrasal(W1,Sem,_,_),
    member(Sem,Semlist).
% removes phrases from first arg that are in nsphrase - lexicon of non-sem.
phrases
remove_no_sem([],[]) :- !.
remove_no_sem([W|Tail],Sfinal) :-
            nosem(W,Phrase), %phrase beg. with W
            begsublist(Phrase,[W|Tail],S1), %remove from sentence
            remove_comma(S1,S2),   %remove "," if it is next
            remove_no_sem(S2,Sfinal), !.
remove_no_sem([W|Tail],Sfinal) :-
            remove_no_sem(Tail,S1),
            append([W],S1,Sfinal) , !.
remove_comma([','|Tail],Tail).
remove_comma(S,S).
% remove_sem(+Sent,-NewSent): Sent is the original sentence, NewSent is
%    stripped of all phrases that are defined in lexicon
remove_sem([],[]) :- !.
remove_sem(S,NewS) :-
    check_sem(S,Rest,_,_),   % phrase in sent. is in lexicon - remove it
    remove_sem(Rest,NewS), !.
remove_sem(S,NewS) :-
    check_no_sem(S,Rest,_),   % phrase in sent. is in nosem list - remove it
    remove_sem(Rest,NewS), !.
remove_sem([X|Tail],[X|NewS]) :-
    remove_sem(Tail,NewS), !.  % not a phrase, process rest
% remove_words(+Sent,-NewSent): Sent is the original sentence, NewSent
%   is stripped of all words that are in lexicon
```

```
remove_words([],[]) :- !.
remove_words([X|Rest],NewRest) :-
     ( (foundword(X); number(X)),     % X is defined in lexicon
       remove_words(Rest,NewRest) ,!;
       remove_words(Rest,New), NewRest = [X|New], !  % X is not in lexicon
     ).
%maxphrase(+ListofPhrases,+Maxin,-MaxOut,InitMaxLen) is true if
%   ListofPhrase is a list of multi-word phrases,
%     Maxin is phrase with maximum words so far
%     MaxOut is phrase with maximum length of phrases in ListofPhrases
%     InitMaxLen is length of initial phrase which is of max. length
maxphrase([],Maxin,Maxin,_) :- !.  % no more phrases - maximum is same as maxin
maxphrase([P|Rest],Maxin,Maxout,InitMaxLen) :-
     length(P,Len),  % length of first phrase
     ( Len > InitMaxLen, !, maxphrase(Rest,P,Maxout,Len);
       Len < InitMaxLen, !, maxphrase(Rest,Maxin,Maxout,InitMaxLen)
     ).

%%%%%%%%%%%%%%%%%%%%%%%%%% lexical interface predicates %%%%%%%%%%%%%%%%%%%%%%
%acclex(Sem,W,S0,S) :-
%   outputform(htext), !, acclex1(Sem,W,S0,S).
acclex(Sem,W,S0,S) :-
   acclex2(Sem,W,S0,S).
acclex(Sem,W,S0,S) :-
   acclexss(Sem,Syn,Target,Features,S0,S).
% check lexicon for word or phrase, Target form is original W
acclex1(p,[P,C],[W|Rest],Rest) :-
         is_list(W),
         find_sem_phrase(p,[P,C],W).
acclex1(p,[P,C],[W|S],S) :- atom(W),
                            wdef(W,p,[P,C]).
acclex1(Sem,[W],[W|Rest],Rest) :-
         is_list(W), %if bracketed list, get Sem and Code from phrasal lexicon
         find_sem_phrase(Sem,_,W).

acclex1(Sem,W,[W|S],S):-   atom(W),
                           wdef(W,Sem,_).
% check lexicon for word or phrase, Target form is taken from lexicon
%acclex2(Sem,Code,[W|Rest],Rest) :-
%         is_list(W), %if bracketed list, get Sem and Code from phrasal lexicon
%         find_sem_phrase(Sem,Code,W).

acclex2(Sem,Code,[W|S],S):-   foundword(W,Sem,Code),
                                          nonvar(Code).    % protect against
lex. error
% find a phrase [W|Tail] in lexicon that begins with W and has category Sem
find_sem_phrase(Sem,Code,[W|Tail]) :-
         phrasal(W,Sem,[W|Tail],Code),  % phrase and code beg. with W
         nonvar(Code).
% case where phrase is already bracketed, look up phrase
sem_finding_phrase1(Code,[W|Tail],Tail) :-
         is_list(W), %phrase is bracketed
         find_sem_sent(Code,W),
          nonvar(Code).    %protect against lexical error
% case where phrase is already bracketed, look up phrase
sem_finding_phrase2(Code,[W|Tail],Tail) :-
         is_list(W), %phrase is bracketed
```

```
          find_sem_sent(Code,W),
           nonvar(Code).     %protect against lexical error
% Phrasal succeeds if lexicon contains phrase
phrasal(W1,Sem,Phrase,Code) :-
       phrase(W1,Sem,Phrase,Code,_).    %multi-word phrase in lexicon
% added March15, 1999
phrasal(W1,Sem,Phrase,Code) :-
            semp(W1,Sem,Phrase,Code,Features).
% lexical definition from marked up input
phrasal(W1,Sem,[W1|Tail],Code) :-
            phr(W1,Sem,Tail,Code).
acclexss(Sem,Syn,Target,Features,[W|S],S):-
            atom(W),
            semw(W,Sem,Target,Features),
            synw(W, Synclass),
            member(Synclass,Syn).
acclexss(Sem,Syn,Target,Features,[W|S],S):-
            is_list(W),
            find_phrasess(W,Sem,Syn,Target,Features).
find_phrasess([W1|Tail],Sem,Syn,Target,Features):-
            semp(W1,Sem,[W1|Tail],Target,Features),
            synp(W1,[W1|Tail],Synclass),
            member(Synclass, Syn).

% lexical definition of a complete finding
find_sem_sent(Code,[W|Tail]) :-
          sem_finding_sent(W,[W|Tail],Code).

listify(C,[C]) :-
          atom(C), !.
listify(C,C) :-
           is_list(C), !.

% distributes left mods and right mods over list of findings creating
% list of lists of findings with mods
distributemods([],[],_,_,_) :- !.
distributemods(Dist,[D1|Tail],Lmods,Rmods,Type) :-
        distributemods(Dist2,Tail,Lmods,Rmods,Type), %distributed for remainder
        mergemods(Lmods,Rmods,Allmods),
        frame(D,Type,D1,Allmods),    %Type frame with mods
        append([D],Dist2,Dist).      % Combine findings to get list of findings

% fixconj - if Leftmods has [certainty,no], and Conj = or, change Conj to and.
%       no A or B = no A and no B; 'denies A,B, or C' is similar.
fixconj(Leftmods,Conj,[rel,and]) :-
        (member([certainty,no],Leftmods); member([certainty,deny],Leftmods)),
        Conj = [rel,or].
fixconj(_,Conj,Conj).

%        write_sentences/1 inputs a PROLOG list and prints out lines
%        which which are English sentences. No wrapping is done.
write_sentences([]) :- !.
write_sentences([X]) :- write(X), nl.  % special sentence - section name
write_sentences(['<',p,'/','>']) :-
     write('<p/>'), nl.    % paragraph mark
write_sentences([X|Rest]) :-
        upper_first([X|Rest],[U|Rest]),
```

```
        write(U),  % First letter of first word made upper case
        %write(X),
        (X = U, chkforpunct(U,Rest), !, write_terms(Rest);  % no space needed
        write(' '), write_terms(Rest)
        ).

%        write_sentence/2 inputs a PROLOG list and prints out an English
%        sentence wrapped. Idlen is the starting position of the sentence
%        in the output.
%         uses libraries ctypes, basic, not
write_sentence([X|Rest],Idlen) :-
    upper_first([X|Rest], [U|Rest]),
    write(U),
    name(U,LU),length(LU,L),
    (U = X, chkforpunct(U,Rest), !, write_terms(Rest, L+Idlen);
     write(' '), write_terms(Rest, L+Idlen+1)
    ).

%       write_list inputs a PROLOG list and prints out a sentence like list.
%        wrapped. Idlen is the starting position of the list in the output.
write_list([X|Rest],Idlen) :-
    write(X),
    name(X,LU),length(LU,L),
   ( chkforpunct(X,Rest), write_terms(Rest, L+Idlen), !;
     write(' '), write_terms(Rest, L+Idlen+1)).
%write_list(+List,+Idlen,-Idlenout)
% write_list prints out a sentence like list with wrapping if necessary.
%   List is the list to be printed
%   Idlen is the column position at start
%   Idlenout is the column position at end
write_list([],Len,Len) :- !.
write_list([X|Rest],Idlen,Idlenout) :-
    atomic(X), write(X),
    name(X,LU), length(LU,L),
    (L + Idlen > 74, nl, Idlen2 = 1, !;
     Idlen2 = L + Idlen, !
    ),
  (chkforpunct(X,Rest), write_list(Rest,Idlen2,Idlenout), !;
    write(' '), write_list(Rest,L+Idlen2+1,Idlenout), !
  );
   is_list(X), write_list(X,Idlen,Idlen2), write_list(Rest,Idlen2,Idlenout).

upper_first([X|Rest], [U|Rest]):-
     name(X, [L|Z]),
 (is_alpha(L), Up is L - 32, !; Up = L),
 name(U,[Up|Z]), !.

% write_terms/1 writes out a word followed by blank, except for punctuations.
write_terms([]) :- !.
% case where X is end of sentence
write_terms([X|Rest]) :-
   (X = '.'; X = ';'),  % last word of sentence
   write(X), nl,  !, write_sentences(Rest), !.
% case where X is interior of sentence
write_terms([X|Rest]) :-
     write(X),
     (chkforpunct(X,Rest),  write_terms(Rest);
```

```
      write(' '), write_terms(Rest)
     ), !.
% write_terms(List,Used): writes the terms in list and counts the number
%       of columns used; starts new line if 75 columns have been used
write_terms([],_) :- !.
% at end of list
write_terms([.], _) :- write('.'), nl,!.
write_terms([;],_) :- write(';'), nl,!.
% X is a punctuation, don't add to final count
write_terms([X|R],Used) :-
  ( R = [], write(' '), write(X), !;
    chkforpunct(X,R),
    write(X), write_terms(R,Used), !
  ).
% X is last word in sentence
write_terms([X,.], Used):-
   name(X, List), length(List, Len),
   Need is Len + 2,
   Total is Used + Need,
   (Total =< 75, write(' '),write(X), write(.);
    Total > 75, nl, write(' '),write(X), write(.)),
   nl, !.
% X is last word in sentence
write_terms([X,;], Used):-
   name(X, List), length(List, Len),
   Need is Len + 2,
   Total is Used + Need,
   (Total =< 75, write(' '),write(X), write(';');
    Total > 75, nl, write(' '),write(X), write(.)),
   nl, !.
% X is followed by ','
write_terms([X,','|Rest], Used):-
   name(X, List), length(List, Len),
   Need is Len + 2,
   Total is Used + Need,
   (Total =< 75, write(' '),write(X), write(','),
    write_terms(Rest, Total);
    Total > 75, nl, write(' '),write(X), write(','),
    New is Need - 1, write_terms(Rest, New)),
    !.
% writes blank + name of X, used is length of name+1
write_terms([X|Rest], Used):-
   name(X, List), length(List, Len),
   Need is Len + 1,
   Total is Used + Need,
   (Total =< 75, write(' '), write(X), write_terms(Rest, Total);
    Total > 75, nl, write(' '),write(X), write_terms(Rest, Len)),!.
write_terms(['X''s'|Rest], Used):-
   name(X, List), length(List, Len),
   Need is Len + 3,
   Total is Used + Need,
   (Total =< 75, write(' '), write(X),write("'s"),
    write_terms(Rest, Total);
    Total > 75, nl, write(X), write_terms(Rest, Len)),!.
% processes sentences in Infile; writes formats to Outfile
% sentences beginning with '%' are treated as comments
testsents(Infile,Outfile) :-
```

```
    see(Infile), seen, see(Infile),
    tell(Outfile),
    readtests,
    see(Infile), seen, told.
% reads next sentence and processes it
readtests :-
    read_in(X),
    (X = end_of_file, !;
     X = [eoff,'.'], !;
     X = [''], !;
     X = ['%'|_], !, readtests;  % don't process comments
     preprocess(X,Bs,Undef,Semlist,skip),
    ( Undef = [],
     dosent(X,Bs,Semlist,Fmt,Message,impression,W,chestxray,strict,0),
     write_sentence(X,1), write(Bs), nl,
     write(Fmt), nl;
     Undef \= [], write_sentence(X,1), write(Bs), nl, write(Undef), nl),
     readtests      % read next sentence
    ).
% Reads in all sentences from input file and creates one list of all sentences
get_inputsents(Prevlist,Toklist) :-
     read_in(X),
     (X = end_of_file, Toklist = Prevlist, !;
      X = [eoff,'.'], Toklist = Prevlist, !;
      X = [''], Toklist = Prevlist, !;
      (last('',X), append(Toklist,[''],X), !;   %remove
       append(Prevlist,X,Newlist),
       get_inputsents(Newlist,Toklist)
      )).

%get_sentence(+A,-B,-C)
% Gets next sentence from input list containing all sentences read in
% Don't end a sentence if "." is preceded by a number and followed by
% a number and unit measure - 1.25 cm, 1.5 cm, .5 cm
% or is followed by a "." which is part of abbreviation
% get_sentence(A,B,C) - A is list of all sentences in report
%                     - B is list containing one sentence
%                     - C is remainder excluding B
% sgml tag for multi-word phrase containing '.' that is not end of sentence
get_sentence(['<',phr|Tail],Sentence,LRest) :-
       enclosedPart(Tail,phr,Between,Rem), % Between beg. part of open phr and
close tag of phr
     append([sem,=,'"',Sem,'"'],MoreAttributes,Between),  %Sem is value of sem
attribute
      (MoreAttributes = ['>'|Phrase], TargetList = Phrase, !;
       MoreAttributes = [t,=,'"'|TargetPlus], % Target terms plus end of phr
       append(TargetList,['"','>'|Phrase],TargetPlus), !  % t attribute followed
by actual phrase
      ),
      Phrase = [W1|Rest],
      append(Phrase,SRest,Sentence),
      concat_atom(TargetList,Target),
      assert(phr(W1,Sem,Rest,Target)),  % assert lex def according to input
        %Phrase = [W1|PRest],
        %abbrev(W1,[W1|PRest],Target,_),
        get_sentence(Rem,SRest,LRest), !.
```

```
% Ignore sentence starting with '%', get next sentence
get_sentence(['%','%'|Rest],Sent,Remainder) :-
     get_sentence(Rest,_,Rem),
     get_sentence(Rem,Sent,Remainder).
get_sentence([X,.,Y,Z|Rest],[X,.],[Y,Z|Rest]) :-   % break up "140. 3+"
     number(X), number(Y), Z = '+', !.  % Y belongs to '+' for new sentence
get_sentence([X,.,Y,Z|Rest],[N|SRest],LRest) :-       % 1.5 cm
      number(X), number(Y),
      %(wdef(Z,unit,_); Z = x),
      Z \= '+',  % break up "140.  3+"
      !,
      name(X,D1), name(.,D2), name(Y,D3), name('E+00',D4),
      append([D1,D2,D3,D4],D), name(N,D), % put number together
      get_sentence([Z|Rest],SRest,LRest).
% common abbrev
get_sentence([X,.|Rest],[X|SRest],LRest) :-     % abbrev ending in "."
% list of common abbreviations seen in reports should not end sentence
   member(X,[vs,dr,cm,mg]), get_sentence(Rest,SRest,LRest), !.
% list of start of names in reports should not end sentence
get_sentence([X,.|Rest],[X|SRest],LRest) :-     % abbrev ending in "."
   member(X,[ms,mr,mrs,dr,st]),
   skipname(Rest,Rest0),    % skip name part
   get_sentence(Rest0,SRest,LRest), !.
% more known abbreviations
get_sentence([W1|Rest],[Rep|SRest],LRest) :-
     abbrevchk([W1|Rest],_,Rem,Rep),  %  abbreviation
     get_sentence(Rem,SRest,LRest), !.
% possible simple xml tag for new paragraph
get_sentence(['<',p,'/','>'|Rest],Sent,Rem) :-  %skip paragraph marker
    get_sentence(Rest,Sent,Rem), !.
% xml tag for sentence '<s>'
get_sentence(['<',s,'>'|Tail],Sentence,Rest) :-
      enclosedPart(Tail,s,Sent,Rest),
       (last('.',Sent), Sentence = Sent, !;   %already has '.'
      append(Sent,[.],Sentence)
       ), !.          %add '.'
get_sentence([.|Rest],[.],Rest) :- !.  %end of a sentence
get_sentence([;|Rest],[;],Rest) :- !.
% interior of sentence
get_sentence([X|Rest],[X|SRest],LRest) :-
                       get_sentence(Rest,SRest,LRest).
get_sentence([],[],[]).   % no more sentences

% abbrevchk(+WordList,-AbList,-RemList,-Target) is true if an abbrev is prefix
%   of WordList, RemList is suffix of WordList (excluding prefix),
%   AbList is prefix consisting of abbreviation
%   and Target is target form of abbreviation
abbrevchk([W1|Rest],AbList,RemList,Target) :-
     abbrev(W1,AbList,Target,Dom),  % abbrev knowledge base indexed by 1st word
     append(AbList,Rem,[W1|Rest]),  % remainder of abbrev. must be in sentence
     (Dom = general, !;    % abbrev. applies to all domains
      domain(Thisrep), Dom = Thisrep, !;  % abbrev. applies to this domain
      is_list(Dom), member(Thisrep,Dom)  % this domain in abbrev. list
     ),
     ( % add back '.' to sentence if it also signals end of sentence
       Rem = [], last('.',AbList), RemList = ['.'], !  %no more words
      ; % words that generally start a new sentence
```

```
        Rem = [W2|_], last('.',AbList),  member(W2,[his,her,he,she,the,this]),
          RemList = ['.'|Rem], !
          ;   % don't add '.' back
        RemList = Rem
      ).
% skipname(+Beglist,-Endlist): skips next word after "mr" or "st"
skipname([],[]) :- ! .
skipname([_,'''',s|Rest],Rest):- !.  % "Luke's"
skipname([o,'''',_|Rest],Rest):- !.  % "O'Grady
skipname([_|Rest],Rest) :- !.

%get_section(+Toklist,-Sents,-Rest,-Section,-Printname,Addno)
% Toklist contains input list; 1st sentence should be a header;
% Sents are all sentences in section; Section is name of section
% Sentences at beg. of Toklist are ignored until a section header is found
get_section([T|Toklist],Sents,Rest,Section,Printname,Addno) :-
       % first sentence should be section header
      get_sentence([T|Toklist],Sentence,RToklist),
      (section_header(Sentence,Rsent,Section,Printname), % Sentence is a section
header
       append(Rsent,RToklist,RToklist2),
       get_sectionsents(RToklist2,Sents,Rest),
       (Addno = 0, !;  % testing if input begins with section header
        Addno = 1, ! , sectno(Sectno), Newno is Sectno + 1,
        retractall(sectno(_)), assert(sectno(Newno))
       ),
       retractall(paragno(_)), assert(paragno(1)),  %1st parag. of section
       retractall(sentno(_)), assert(sentno(0))    %1st sentence of parag.
       ;   % 1st sentence is not a legitimate header - return []
        Section = []
       % get_section(RToklist,Sents,Rest,Section) % skip till find header
      ), !.

get_section([],[],[],[],_,_).
get_sectionsents([],[],[]) :-!.
get_sectionsents(Toklist,Slist,Rest) :-
     get_sentence(Toklist,Sentence,RToklist),  % one sentence
     (\+ section_header(Sentence,_,_,_), %more sentences in section
        get_sectionsents(RToklist,RSents,Rest),
        append(Sentence,RSents,Slist)
       ; % the next section is a section header - return
      Rest = Toklist, Slist = []).

section_header(S,RestS,'report clinical information item',
          'CLINICAL INFORMATION:.') :-
    (S = [clinical,information,':','.'], !, RestS = [];
     begsublist([clinical,information,':'],S,RestS), !;
     S = [clininfo,':','.'], RestS = [], ! ;
     begsublist([clininfo,':'],S,RestS), !
    ).
section_header(S,RestS,'report impression item',
            'IMPRESSION:.') :-
   (S = [impression,':',.], RestS = [], !;
    begsublist([impression,':'],S,RestS), !
   ).
section_header(S,Rest,'report summary item','SUMMARY:.') :-
    S = [summary,':'|Rest].
```

```
section_header(S,RestS,'report description item','DESCRIPTION:.') :-
   (S = [description,':',.], RestS = [], !;
    begsublist([description,':'],S,RestS), !
   ).
section_header(S,Rest,'report diagnosis item','DISCHARGE DIAGNOSIS:.') :-
   (S = [discharge,diagnosis,':'|Rest] ;
    S = [final,diagnosis,':'|Rest];
    S = [principle,diagnosis,':'|Rest]; S = [associated,diagnosis,':'|Rest];
    S = [transfer,diagnosis,':'|Rest];
    S = [diagnosis,'(',es,')',':'|Rest];
    S = [diagnosis,:|Rest]
   ), !.
section_header(S,Rest,'report laboratory data item','LAB DATA:.') :-
    S = [laboratory,data,':'|Rest], !.
section_header(S,Rest,'report medications item','MEDICATIONS:.') :-
    S = [medications,':'|Rest], !.
section_header(S,Rest,'report current medications item','MEDICATIONS:.') :-
    S = [current,medications,':'|Rest], !.
section_header(S,Rest,'report discharge medications item',
        'DISCHARGE MEDICATIONS:.') :-
    S = [discharge,medications,':'|Rest], !.
section_header(S,Rest,'report discharge disposition item',
     'DISCHARGE DISPOSITION:.') :-
    S = [discharge,disposition,':'|Rest], !.
section_header(S,Rest,'report medications on admission item',
     'MEDICATIONS:.') :-
    S = [medications,on,admission,':'|Rest], !.
section_header(S,Rest,'report medications on transfer iterm',
     'MEDICATIONS:.') :-
     S = [medications,on,transfer,':'|Rest], !.
section_header(S,Rest,'report procedure item','PROCEDURE:.') :-
  (S = [operation,':'|Rest]; S = [procedure,':'|Rest]
  ), !.

section_header(S,Rest,'report indications for procedure item','INDICATIONS:.')
:-
  (S = [indications,for,procedure,':'|Rest]; S =
[indications,for,operation,':'|Rest]
  ),
   !.

section_header(S,Rest,'report preoperative diagnosis item','PREOP DIAGNOSIS:.')
:-
   S = [preoperative,diagnosis,':'|Rest], !.
section_header(S,Rest,'report admitting diagnosis item','ADMITTING
DIAGNOSIS:.'):-
   S = [admitting,diagnosis,':'|Rest], !.
section_header(S,Rest,'report postoperative diagnosis item','DIAGNOSIS:.') :-
   S = [postoperative,diagnosis,':'|Rest], !.
section_header(S,Rest,'report physical examination item',
        'PHYSICAL EXAM:.') :-
   S = [physical,examination,':'|Rest], !.
section_header(S,Rest,'report chief complaint item','CHIEF COMPLAINT:.') :-
   S = [chief,complaint,':'|Rest], !.
section_header(S,Rest,'report hospital course item','HOSPITAL COURSE:.') :-
   S = [hospital,course,':'|Rest], !.
```

```
section_header(S,Rest,'report allergy item','ALLERGIES:.') :-
   S = [allergies,':'|Rest], !.

section_header(S,Rest,'report follow up item','FOLLOW UP:.') :-
   S = [follow,up,':'|Rest], !.
section_header(S,Rest,'report findings item','FINDINGS:.') :-
   S = [findings,':'|Rest], !.
section_header(S,Rest,'report indications and findings item','FINDINGS:.') :-
   S = [indications,and,findings,':'|Rest], !.
section_header(S,Rest,'report indications and findings item','INDICATIONS:.') :-
   S = [indications,':'|Rest], !.
section_header(S,Rest,'report provisional diagnosis item','PRELIM DIAGNOSIS:.')
:-
   S = [provisional,diagnosis,':'|Rest], !.
section_header(S,Rest,'report review of systems item','REVIEW OF SYSTEMS:.') :-
   S = [review,of,systems,':'|Rest], !.
section_header(S,Rest,'report past history item','PAST MEDICAL HISTORY:.') :-
   S = [past,history,section,':'|Rest], !.
section_header(S,Rest,'report past history item','PAST MEDICAL HISTORY:.') :-
   S = [past,medical,history,':'|Rest], !.
section_header(S,Rest,'report social history item','SOCIAL HISTORY:.') :-
   S = [social,history,':'|Rest], !.
section_header(S,Rest,'report past history item','PAST MEDICAL HISTORY:.') :-
   S = [history,':'|Rest], !.
section_header(S,Rest,'report past history item','PAST MEDICAL HISTORY:.') :-
   S = [brief,history,':'|Rest], !.
section_header(S,Rest,'report history of present illness item',
         'HISTORY OF PRESENT ILLNESS:.') :-
   S = [history,of,present,illness,':'|Rest], !.
section_header(S,Rest,'report history of present illness item',
         'HISTORY OF PRESENT ILLNESS:.') :-
   S = [history,of,the,present,illness,':'|Rest], !.
section_header(S,Rest,'report specimen item','SPECIMEN') :-
   S = [specimen|Rest], !.

% sentence consists of id number only or "." only.
isidentifier([X,.]) :-
        integer(X).
isidentifier([X,;]) :-
        integer(X).
isidentifier([.]) :- !.  % sentence consists only of .
isidentifier(['.','<eos>']) :- !.
isidentifier(['<',p,'/','>']) :-   % paragraph marker sentence - update no.
       paragno(N),
       retractall(paragno(_)),
       Newno is N + 1,
       assert(paragno(Newno)),
       retractall(sentno(_)),
       assert(sentno(0)).

% skipsentence is true, if sentence should be ignored.
% Skip sentences containing family info
skipsentence([X|_]) :-
   foundword(X,family), !.
skipsentence([X|_]) :-
   foundword(X,insurance), !.
% This occurs if sentence contains
```

```
% a sequence in skips database and sentence also contains findings.
skipsentence([X|Rest],Semlist,Error) :-
   skips([X|Sseq]),    % X is the beg. of subseq. in skip database
   prefix([X|Rest],[X|Sseq]), % sentence contains subseq.
   (subtype(_,Semlist),  % sentence contains information to be extracted
    Error = no;  % don't try to segment
    Error = yes), !.   % treat sentence as error and try to segment.

skipsentence([_|Rest],Semlist,Error) :-
   skipsentence(Rest,Semlist,Error).

% findingseg(+S,-Fseg,-Begseg): partitions sentence
%       S is the sentence; Begseg is the segment preceding the
%         modifiers of the finding; Fseg is the segment of S starting
%         with the leftmost modifier of the finding and consists of the
%         remaining sentence.
findingseg(S,Fseg,Begseg) :-
    partition(S,Begpart,Restpart),
    (Begpart = [], Begseg = [];
     Restpart = [], Fseg = [], Begseg = S;
    right1stmod(Begpart,Begseg,Modseg)),
    append(Modseg,Restpart,Fseg).
findingseg(_,[],_) :- !.
actionfindingseg(S,Fseg,Begseg):-
      partition(S,Begpart,Restpart),
    (Begpart = [], Begseg = [];
     Restpart = [], Fseg = [], Begseg = S;
      reverse(Begpart, ReversedBefore),
          findsubstance(ReversedBefore,Rest),
          append(Substancepart,Rest,ReversedBefore),
          reverse(Substancepart,Leftpart),
        reverse(Rest,Begseg),
      append(Leftpart,Restpart,Fseg)).
actionfindingseg(_,[],_) :- !.
findsubstance([],[]):- !.
findsubstance([X|Rest],Rest):-
      substance(_,[X],[]),!.
findsubstance([X|Rest1],Rest):-
      findsubstance(Rest1,Rest).

% partition(+S,-Begpart,-Restpart): partitions sentence
%       S is initial
% partition(+S,-Begpart,-Restpart): partitions sentence
%       S is initial sentence; Begpart is part of sentence before the
%         finding; Restpart is the rest of the sentence and starts with
%         the finding. If there are 2 consecutive findings
%         the 1st one is considered a modifier
partition([],[],[]) :- !.
partition([X|Rest],[X|Begpart],Restpart) :-
    not(isfinding(X)), !, partition(Rest,Begpart,Restpart).
partition([X,Y|Rest],[X],[Y|Rest]) :-
    isfinding(X), isfinding(Y), !.
partition([X|Rest],[],[X|Rest]) :-
    isfinding(X), !.

% isfinding(+X): is true if X is a word or phrase whose semantic class
%       is a finding or subtype of finding.
```

```
isfinding(X) :-
     foundword(X,Sem),    % semantic class of word
     subtype(_,[Sem]).     % is class a type of finding, recommend, or technique
% semantic class which are types of relevant information
subtype(finding,Sem) :-
     intersect(Sem,[attach, createbond, breakbond,activate,
      inactivate,substitute,transcribe,express,promote,
      signal]).
% there is only one type of technique class
subtype(technique,Sem) :-
     member(technique,Sem).
subtype(time,Sem) :-
     intersect(Sem,[status,sstatus,change,tmper,vstatus]).
findinginlist(Sem) :-
    intersect(Sem, [attach, createbond, breakbond,activate,
      inactivate,substitute,transcribe,express,promote,
      signal]).

% chkforpunct(+W,+Rest): is true if there should be no space after word W
chkforpunct(W,_) :- member(W,['/','<','>','-','"','[',']',
                '{','}','_','+','=','|','\']), !.
% nothing left to write.
chkforpunct(W,[]) :-!.
% is true if there should be no space before word after current word
chkforpunct(_,[W|_]) :-
   ispunct(W).
% ispunct(+W) is true if W is a punctuation for sentence print out
%  The following characters are not treated as punct: ~ ` # $ ^ & *
ispunct(W) :- member(W,[',','.',';','/','<','>','?','''','-',':','"','[',']',
                '{','}','(',')','_','+','=','|','\','%','@']).
% right1stmod(List,Firstpart,Modpart): Modpart begins with the first
%   word in List which is a modifier; Firstpart are the preceding words
right1stmod([],[],[]) :- !.
% X is a modifier or finding; Beginning part is empty
right1stmod([X|Rest],[],[X|Rest]) :-
   foundword(X,Sem,Target),
   (modifier(Sem); Sem = p, Target = [_,conn]; subtype(_,[Sem])), !.
% X is not a modifier or finding
right1stmod([X|Rest],[X|Firstpart],Modpart) :-
   right1stmod(Rest,Firstpart,Modpart).

% frame(Frame,Type,Value,Mods): creates a list Frame, whose 1st
%       element is Type, 2nd element is Value, and 3rd is a list of
%       modifier frames or is emtpy
% Case where modifier list is empty; Value should be atom except for
% certain types;
frame([Type,Value],Type,Value,X) :-
    (X = []; X = [[]]),
    atom(Value), !.
% Special cases where value of type should be a list
frame([Type,[H|R]],Type,[H|R],X) :-
       (X = []; X = [[]]),
       oklist(Type), !.
% Modifier list is merged with list consisting of Type and Value
frame(Frame,Type,Value,Mods) :-
     atom(Value),
     append([Type,Value],Mods,Frame), !.
```

```
frame(Frame,Type, [H|R], Mods):-
     is_list(R),
     append(R, Mods, NewMods),
     append([Type, H], NewMods, Frame), !.
% Components of Frame
frame([Type,Value|Mods],Type,Value,Mods) :- !.
% Value of Type should not be a list; first element of value is real value
frame([Type,H,Rest],Type,[H|Rest],[]) :- !.
% Special cases where value of type should be a list
%frame([Type,[H|R]],Type,[H|R],[]) :-   %repeated from rule above
%   oklist(Type), !.
% Value of Type should not be a list; first element of value is real value
frame(Frame,Type,[H|Rest],Mods) :-
    mergemods(Rest,Mods,NewMods),
    append([Type,H],NewMods,Frame).

% mergemodinf(-F,+Frame,+Mods): Frame is a type-value-mod frame; Mods
%   is an additional set of modifiers for Frame; mergemodinf adds Mods
%   to Frame, resulting in F.
mergemodinf([],[],_):-!.
mergemodinf(F,[rel,X|Rest],Modrel):-
        mergemodinf(F1,Rest,Modrel),
        append([rel,X],F1,F),!.
mergemodinf(F,[F1,X|Modfin],Modrel):-
        atom(F1),mergemods(Modrel, Modfin, Mod),
        append([F1,X],Mod,F),!.
mergemodinf(F,[H|R],Modrel):-
        mergemodinf(F1,H,Modrel),
        mergemodinf(F2,R,Modrel),
        append([F1],F2,F).
% addmodstof(+Args,+Mods,-NewArgs) is true if Args is a list of formats,
% Mods is a list of modifiers and NewArgs is a list of formats where Mods
% has been added to modifier list of that format
addmodstof([],_,[]) :- !.   % no more formats
addmodstof([Format1|Rest],Mods,[F1|NewRest]) :-
       mergemodinf(F1,Format1,Mods),  % merge modifiers into 1st format
       addmodstof(Rest,Mods,NewRest), !.  %add modifier to remaining
% oklist(+Type): is true if Type can have a list as its value
oklist(unitval).
oklist(age).
oklist(measure).
oklist(prev_timeunit).
oklist(future_exam).

% mergemods(+Mods1,+Mods2,-Mod): Mods1 and Mods2 are a list of modifier lists
%       Mod is the merged list; some elements of Mods1 and Mods2 may be
%       empty
mergemods([],M,M) :- !.
mergemods(M,[],M).
mergemods(Mods1,Mods2,Mod) :-
        delete(Mods1,[],M1),
        delete(Mods2,[],M2),
        append(M1,M2,Mod).

% addmod(+Mod,+Modlist,-NewMod): NewMod is formed by including
%       Mod into Modlist
addmod([],Mod,Mod) :-!.
```

```
addmod(Mod,[],[Mod]) :- !.
addmod(Mod,Modlist,NewMod) :-
   append([Mod],Modlist,NewMod).
% modlist(+ListofMods,-Mods): ListofMods is a list consisting of
%   individual modifier frames, some of which may be empty
%   Mods is formed as a list of non-empty modifiers
modlist([],[]) :- !.
% ignore a modifier which is an empty list
modlist([[]|R],Mods) :-
    modlist(R,Mods), !.
modlist([[H|R1]|R2],Mods) :-
    atom(H), !,
    modlist(R2,Rmods),
    addmod([H|R1],Rmods,Mods).
modlist([[H|R1]|R2],Mods) :-
    is_list(H), !, % is first element is a list
    modlist(R2,Rmods),
    mergemods([H|R1],Rmods,Mods).

%bpframe: creates from for sequences of bodyloc/region/position
bpframe(F,[],_,F,[]):- !.  % only 1 bodyloc
bpframe(F,[],Type,Bp1,Bp2) :- % no conj relation but more than 1 bodyloc
        frame(Bp1,Bp1Type,Bp1Val,Bp1Mods),  %contents of Bp1 frame
        frame(Bp2,Bp2Type,Bp2Val,Bp2Mods),  %contents of Bp2 frame
        ( (Bp1Type = region; Bp1Type = position),
         Bp2Type = bodyloc, % 'left lung', 'area of lung'
         mergemods(Bp1Mods,Bp2Mods,BpMods),  %new region modifier
         frame(NewBp2Mods,Bp1Type,Bp1Val,BpMods), %new Bp1 frame w new mod
         frame(F,Bp2Type,Bp2Val,[NewBp2Mods])   % main frame is bodyloc
          ;
         Bp1Type = bodyloc, Bp2Type = bodyloc, Type = main, %Bp2 is main
         mergemods(Bp1Mods,Bp2Mods,BpMods),  %new bodyloc modifier
         frame(NewBp2Mods,Bp1Type,Bp1Val,BpMods), % 'joint of shoulder'
         frame(F,Bp2Type,Bp2Val,[NewBp2Mods])       % main bp frame is shoulder
          ;
         mergemods(Bp1Mods,Bp2Mods,BpMods),
         frame(NewBp1Mods,Bp2Type,Bp2Val,BpMods), % 'shoulder joint'
         frame(F,Bp1Type,Bp1Val,[NewBp1Mods])       % main bp frame is shoulder
        ), !.
bpframe(F,Rel,_,Bp1,Bp2) :-  % no conj relation but more than 1 bodyloc
        Rel = [rel,Conj|_], Bp2 \= [],
        mergemods([Bp1],[Bp2],Conjargs),
        frame(F,rel,Conj,Conjargs).

getrelation(R,F1,F2,F) :-
        (F2 \= [],
            (F1 = [rel,Conj1|Rest1], R = [rel,Conj],
                                (Conj1 = ','; Conj1 = or; Conj1 = and),
                                (Conj = ','; Conj = or; Conj = and);
              Rest1 = [F1]),
            (F2 = [rel,Conj2|Rest2],
                                (Conj2 = ','; Conj2 = or; Conj2 = and);
              Rest2 = [F2]),
            %splice([R,Rest1,Rest2],F);
             append([R,Rest1,Rest2],F);
          F2 = [], F = F1 ).
```

```
uptotal :-
   addstotal(X),
   X =< 50,
   NewX is X + 1,
   retractall(addstotal(X)),
   assert(addstotal(NewX)), !.
```

Appendix E

```
$save{'a'}='AAAC';
$save{'b'}='AAAG';
$save{'c'}='AAAT';
$save{'d'}='AACC';
$save{'e'}='AACG';
$save{'f'}='AACT';
$save{'g'}='AAGC';
$save{'h'}='AAGG';
$save{'i'}='AAGT';
$save{'j'}='AATC';
$save{'k'}='AATG';
$save{'l'}='AATT';
$save{'m'}='ACAC';
$save{'n'}='ACAG';
$save{'o'}='ACAT';
$save{'p'}='ACCC';
$save{'q'}='ACCG';
$save{'r'}='ACCT';
$save{'s'}='ACGC';
$save{'t'}='ACGG';
$save{'u'}='ACGT';
$save{'v'}='ACTC';
$save{'w'}='ACTG';
$save{'x'}='ACTT';
$save{'y'}='AGAG';
$save{'z'}='AGAT';
$save{'0'}='AGCC';
$save{'1'}='AGCG';
$save{'2'}='AGCT';
$save{'3'}='AGGC';
$save{'4'}='AGGG';
$save{'5'}='AGGT';
$save{'6'}='AGTC';
$save{'7'}='AGTG';
$save{'8'}='AGTT';
$save{'9'}='ATAT';
$save{' '}='ATCC';
$save{']'}='ATCC';
$save{'['}='ATCC';
$save{';'}='ATCC';
$save{':'}='ATCC';
$save{'"'}='ATCC';
$save{'\''}='ATTC';
$save{'?'}='ATCC';
$save{'!'}='ATCC';
$save{'#'}='CCCG';
$save{'$'}='CCCT';
$save{'^'}='CCGG';
$save{'&'}='CCGT';
$save{'*'}='CCTG';
$save{'('}='ATCC';
$save{')'}='ATCC';
```

```
$save{'_'}='CGCT';
$save{'-'}='ATCC';
$save{'+'}='CGGT';
$save{'='}='CGTG';
$save{'}'}='CGTT';
$save{'{'}='CTCT';
$save{','}='ATCC';
$save{'.'}='ATCC';
$save{'|'}='CTTG';
$save{'%'}='CTTT';
$save{'/'}='ATCC';
$save{'\\'}='GGTT';
$save{'@'}='GTGT';
$save{"\n"}='ATCC';
$save{'<'}='GTTT';
$save{'>'}='GTTT';
$save{'~'}='GTTT';
```

Appendix F

```
#!/usr/bin/perl
#Scan.pl : Scans blast output
#Author: Michael Krauthammer
#Copyright: c.1999, Columbia University

#Variables

#blast input/file
$input_file="genebank.result";
#program output
$output_file="match.txt";

#open datastream for file which contains blast output
    open (INPUT,'/storage/psi-blast/MarkIt/programs/markIt.result');

while ($line=<INPUT>){
    if ($line=~/\>gi\|(\d*) (.*)\,(.*)\,(.*)/){
   $target=$4;
   $gi =$1;
   $semantic_class=$3;
    }
    if($line=~/Length = (.*)/){
   $lengthI=$1;
    }
    if ($line=~/Identities \= (\d*)\//){
   $length_actual=$1
    }
    if ($line=~/Query: (\d*)/){
   $start=$1;
    }
#print if Subj 1, sometimes match 2 or 3 line long

    if ($line=~/Sbjct: 1 /){
   if (($length_actual/$lengthI) > .9){
   print
$target,"|",$start,"|",$start+$lengthI,"|",$semantic_class,"|",$gi,"\n";
    }
    }
}
```

Appendix G

```
#!/usr/bin/perl
#nucleotide_text_parser.pl
#Author: Michael Krauthammer, c.1999 Columbia University

open (INPUT,$ARGV[0]);

#read uncoded input text line by line (chop it)
$all='';
while ($line=<INPUT>){
    $all=$all.$line;
   }
open (INPUTII,'/storage/psi-blast/MarkIt/programs/markItII.result');
open (OUTPUT,'>result.txt');
#first part: check matches, store positions

while ($line=<INPUTII>){

($name,$start,$end,$semantic_class,$gi)=$line=~/(.*)\|(.*)\|(.*)\|(.*)\|(.*)/;

#divide by 4 (4 letter code)
$start=($start-1)/4;
$end=($end-1)/4;

#get substring
if ($start != 0){
$letters=substr($all,$start-1,$end-$start+3)."|";
} else {
$letters = ' '.substr($all,0,$end+2)."|";
}
($letter_beginning)=$letters=~/(^.)/;
$letter_end=substr($all,$end,1);
$letter_endII=substr($all,$end,2);
#ignore matches that are in the MIDDLE of sentences, allow plurals
$letter_beginning=~tr/[A-Z]/[a-z]/;
$letter_end=~tr/[A-Z]/[a-z]/;
if ((!($letter_beginning=~/[a-z]/)) && ((!($letter_end=~/[a-z]/)) ||
($letter_endII=~/s /))){

#make sure only the first occurence is stored at  this position
   if ($save{$start}==''){
   $save{$start}=$end.'|'.$semantic_class.'|'.$gi;
   }
         foreach $key(keys(%save)){
   ($end_key)=$save{$key}=~/^(.*)\|/;
   if ($end_key>$end){
      if ($key<$start){
         $save{$start}='null',
       }
   }
    }
}
}
```

```
#second part: print out marked up document
sort(%save);
for ($i=0;$i<length($all);$i++){
    if ((!$save{$i}=='null') && ($save{$i}=~/./)){
   ($end,$semantic_class)=$save{$i}=~/(.*)\|(.*)\|/;
   print OUTPUT '<phr="',$semantic_class,'">';
   $store=substr($all,$i,$end-$i);
   print OUTPUT $store;
   print OUTPUT "</phr>";
   $i=$end-1;
    } else {
   $store=substr($all,$i,1);
   print OUTPUT $store;

    }
}
```